Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra měřicí a řídicí techniky

Návrh architektury biomedicínského systému pro domácí péči a jeho testování

Architecture Design of Biomedical System for Personal Homecare and his Testing

2009 **Motalová Leona**

Prohlášení

Prohlašuji, že

- jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.
- jsem byla seznámena s tím, že se na mou diplomovou práci plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména §35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a §60 – školní dílo.
- beru na vědomí, že Vysoká škola báňská – technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně ke své vnitřní potřebě diplomovou práci užít (§35 ods. 3).
- souhlasím s tím, že jeden výtisk diplomové práce bude uložen v Ústřední knihovně VŠB-TUO k prezenčnímu nahlédnutí a údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO.
- beru na vědomí, že odevzdáním své práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, bez ohledu na výsledek její obhajoby.

V Ostravě 7. 5. 2009

……

Poděkování

Tímto bych chtěla poděkovat vedoucímu mé diplomové práce Ing. Ondřeji Krejcarovi PhD. za vedení, pomoc a cenné rady, při vypracování této práce .
Dále bych chtěla poděkovat panu doc. Ing. Jindřichu Černohorskému PhD. Za poskytnuté odborné rady a konzultace.

Abstrakt

Tato diplomová práce se zabývá analýzou informačního systému pro správu dat a plánování úkonů agentury domácí péče, jeho návrhem a testováním. Vyvíjená aplikace má ulehčit zaměstnancům práci a v propojení s mobilní aplikací, umožnit zaměstnancům zobrazovat a spravovat své úkoly přímo v pracovním terénu, popřípadě zadat krátkou poznámku k některému z pacientů.

Součástí této práce je zpracování veškerých požadavků do podrobné systémové specifikace, která udává, jak má aplikace fungovat, jaké má vlastnosti a omezení nebo také, jak má vypadat uživatelské rozhraní aplikace. Systém je analyzován UML diagramy do takové míry, aby byl dostatečně popsán a byly pokryty veškeré funkce systému a způsob jejich práce. Systém jako celek je zobrazen v Use Case diagramu a rozepsán do jednotlivých tříd a jejich vlastností v diagramu tříd. Práce jednotlivých případů užití je pak zobrazena v diagramech aktivit a sekvenčních diagramech.

Nakonec je systém testován a to převážně ze zátěžového hlediska nebo testy uživatelského rozhraní. Také je v této části ověřováno, zda systém vyhovuje systémové specifikaci.
Aplikace je navržena tak, aby ji bylo možno jednoduše rozšířit nebo modifikovat k jiným účelům. Udává tak určitý rámcový vzor pro různé systémy správy.

Abstract

This thesis deals with the analysis of the informational system for the administration of the datas and the projecting of the duties of the home care agency, its proposal and testing. The developing application should facilitate to employees their work and in the interconnection with the mobile application to provide them to display and to operate their duties directly in the work ground ,eventually to input a short note to some patients.

A part of this work is the processing of all requests to the detail systemic specification which determinates how the application has to work, how it has qualities and the limitation or also how should appear the custom interface of the application. The system is analyzed by the UML diagrams to such measure so that to be enough described and so that there were covered all functions of the system and the method of their work. The system as the unit is displayed in the Use Case diagram and itemized to the particular categories and their qualities in the diagram of the sorts. The work of the particular events of the utilization is then displayed in the diagrams of the activities and in the sequential diagrams.

The system is finally tested and so mainly from the burdening aspect or by the tests of the custom interface. In this part is also checked if the system conforms to the systemic specification.The application is proposed so that it was possible to extend it simply or to modify to the different intentions, so it denotes certain general model for the different systems of the administration.

Klíčová slova

.NET, SQL databáze, Microsoft Visual Studio 2008, C#, životní cyklus vyvíjeného systému, analýza systému, návrh systému, architektura systému, implementace, testování softwaru, systémové požadavky, systémová specifikace, analytický diagram, strukturovaná analýza, objektově orientovaná analýza, časový plán, návrhové vzory, datový model, grafické uživatelské rozhraní, zátěžové testování, relace.

Key words

.NET, SQL database, Microsoft Visual Studio 2008, C#, System development life cycle, system analysis, software design, system architecture, implementation, software testing, system requirements, system specification, analytic diagram, structured analysis, object oriented analysis, time plan, design patterns, data model, graphical user interface, stress testing, relation.

Seznam použitých symbolů a zkratek

.NET – technologie firmy Microsoft
DFD – Data Flow Diagram
GUI – Grafical User Interface (grafické uživatelské rozhraní)
IIS – Internet Information Services (Internetová Informační Služba)
LINQ – Language INtegration Query (integrovaný jazyk pro dotazování)
MVC – Model View Controller
OS – Operační Systém
PC – Personal Computer (Osobní počítač)
RAM – Random Access Memory (paměť s náhodným přístupem)
SQL – Structured Query Language (strukturovaný dotazovací jazyk)
TCP/IP – Transmission Control Protocol/Internet Protocol (Internetový protokol pro přenos dat/internetový protokol)
UML – Unified Model Language
Wi-Fi – Wireless-Fidelity (bezdrátový - spolehlivý)
WPF – Windows Presentation Foundation

Obsah

1 Úvod

Tato diplomová práce se zabývá analýzou, návrhem a v neposlední řadě testováním aplikace spravující data pro agenturu domácí péče. Vývoj této aplikace je řešen v diplomové práci pana Dalibora Janckulíka. Tyto dvě práce na sebe tedy navazují a jsou vzájemně propojeny.

Cílem vyvíjeného systému je modernizace stávajícího řešení správy dat pacientů a zaměstnanců a plánování úkonů pacientům, z dnešní formy postavené na databázi Microsoft Access, do modernější verze, postavené na databázi Microsoft SQL Server 2008, která by zdravotníkům jejich práci ulehčila.

Při zahájení práce na tomto systému bylo nejprve zapotřebí podrobně sepsat veškeré požadavky na systém do přehledné, přesné a hlavně jednoznačné systémové specifikace. Ta udává, jak má systém fungovat, jaké bude mít funkce a omezení, dána dostupnými technologiemi nebo i požadavkem uživatelů. Jedním z hlavních požadavků agentury bylo, aby systém plně pracoval v offline režimu, a to z důvodu spravování choulostivých osobních dat pacientů.

Analýza pomocí systémové specifikace je doplněna o objektově orientovanou analýzu, tedy o UML (Unified Model Language) diagramy. Ty jsou vhodné převážně pro správné pochopení funkčnosti a způsobu práce aplikace popsané v systémové specifikaci a to především vývojářem. Pro tuto práci byly zvoleny diagramy případů užití, diagram tříd, sekvenční diagramy a aktivity diagramy. Těmito diagramy bylo popsáno vše potřebné, a proto další typy nebylo třeba použít.

Při návrhu aplikace je nutné navrhnout databázi, do které budou veškerá data ukládána, a to tak, aby databáze neobsahovala duplikované údaje. Proto je navržena tak, aby veškerá data byla uložena pouze v jedné tabulce, a ostatní tabulky, se v případě potřeby, na tyto data pouze odkazovaly primárním klíčem. Primární klíč je unikátní kód, sloužící k identifikaci zdrojových dat. Celkově navržená databáze obsahuje 20 tabulek. Jejich data a relace jsou podrobněji popsány v této diplomové práci.

Celý systém je nakonec testován v první řadě z pohledu na specifikaci, zda vyvíjená aplikace odpovídá požadavkům zadavatele, v druhé řadě dle navržených scénářů použití, a nakonec po zátěžové stránce, měřením odezvy na požadavek uživatele v závislosti na množství simultánních přístupů k databázi.

Diplomová práce je rozvržena následovně:

Kapitola 2 Životní cyklus systému – popisuje jednotlivé fáze vývoje informačního systému, modely způsobu vývoje životního cyklu systému a vliv použitého modelu na testování systému.

Kapitola 3 Systémové požadavky – tato kapitola pojednává o účelu zjišťování systémových požadavků, popisuje jejich základní typy a vlastnosti.

Kapitola 4 Systémová specifikace – obsahuje obecnou šablonu pro psaní systémových specifikací a její implementaci na vyvíjený systém, podle které je aplikace programována.

Kapitola 5 Analýza systému – v této kapitole je uveden základní popis strukturované analýzy a její použití na vyvíjený systém. Dále podrobněji popisuje objektově orientovanou analýzu, základní typy UML diagramů, jejich prvky a způsob použití. Dále jsou zde popsány diagramy popisující funkčnost a strukturu vyvíjené aplikace pro agenturu domácí péče.

Kapitola 6 Časový plán vývoje informačního systému – zde je rozvržen plán prací do jednotlivých iterací a časů, ve kterých mají být dokončeny a započaty další práce následující iterace vývoje systému.

Kapitola 7 Návrh systému – tato kapitola obsahuje jednotlivé části návrhu systému, od návrhu jeho architektury, databáze a způsobu přístupu systému k jejím datům, až po návrh vzhledu uživatelského rozhraní.

Kapitola 8 Úvod do testování systému – popisuje základní rozdělení a typy testů. Dále obsahuje scénáře použití systému uživateli, popis systému vyvinutého pro zátěžová testování systému a rozbor získaných výsledků testování.

2 Životní cyklus systému

Dříve než je jakýkoliv softwarový produkt uveden na trh, projde několika stupni vývoje. Tento proces je nazýván životním cyklem sytému neboli životním cyklem softwarového produktu.

Životní cyklus systémů nastává po vyřešení základních otázek plánování, návrhu a řízení projektu.

Životní cyklus je tvořen těmito fázemi:

- předběžná analýza (specifikace cílů);
- analýza (identifikace požadavků a definice – specifikace požadavků);
- návrh (projektová studie);
- implementace;
- testování;
- zavádění systému;
- zkušební provoz;
- rutinní provoz a údržba;
- reengineering.

[1], [2]

2.1 Popis fází životního cyklu systému

Každá z následujících fází má svou charakteristiku a udává, co by se v rámci ní mělo s vyvíjeným systémem dít a také má daný specifický cíl, kterého by mělo být dosaženo a tím daná fáze ukončena. V praxi však tyto fáze nenastupují jedna po ukončení druhé, ale často se navzájem prolínají a nelze je od sebe přesně oddělit (například analýza a návrh).

2.1.1 Předběžná analýza

Základem celkového návrhu, vývoje i jakékoli úpravy stávajícího systému jsou požadavky uživatelů a cíle organizace. V této části je nutné shromáždit veškeré požadavky dané zadavatelem, v hrubých rysech rozebrat a odhadnout dobu realizace a náklady na výsledný produkt. Cílem této fáze je sestavit základní seznam požadavků, cílů a funkcí systému.

Celkový rámcový projekt by měl obsahovat následující dokumentace:

- Časový plán projektu.
- Zdroje nutné k řešení, čímž je míněno finance, personál, SW, HW a podobně.
- Odhad funkčnosti, rozsahu systému, ekonomické efektivnosti a návratnosti investice.

Nástroje pro vytvoření specifikačního projektu:

- Analýza současného stavu, cílem je zjistit současný stav, nedostatky a navrhnout změny. (nejedná – li se o nový produkt)
- Získání požadavků uživatelů a zjištění požadovaných vstupních a výstupních informací.
- Seznam problémů, které jsou známy, popis jejich důsledků a nástin řešení.

Konečným dokumentem této části je dokument, který specifikuje účel systému, identifikuje jeho uživatele a jejich zásadní požadavky, definuje části systému a navrhuje jejich řešení, obsahuje seznamy událostí a odhady datových základů, technického a softwarového zajištění.

2.1.2 Analýza

Tato část cyklu je rozborem části předchozí. Její důležitost je klíčová, neboť veškeré chyby ve struktuře dat i systému, které se zde neodhalí, jsou později velice obtížně odstranitelné.

Cílem analýzy je poskytnout úplný, bezrozporný popis systému, který je srozumitelný všem zúčastněným, posouditelný různými zainteresovanými stranami a také testovatelný. Výsledkem analýzy je definice funkčních požadavků. Tato definice se nazývá systémová specifikace, kde jedna funkce označuje jedno, z vnějšku viditelné a testovatelné, chování softwaru.

2.1.3 Návrh

Během návrhu se vytváří logický model systému, včetně uplatnění omezení daných např. existující výpočetní technikou, dostupností programovacího jazyka, atd. Cílem návrhu je vyhovět všem specifikací definovaným požadavkům a generovat podklad pro implementaci, což je výsledkem této části. Tímto výsledkem je dokument, který je podkladem pro obsah smlouvy s externí firmou o návrhu a realizaci informačního systému (IS), časový plán, cena vyvíjeného projektu, konkrétní implementace systému, podmínky zavádění v organizaci, záruční servis a podmínky celkového předání IS.

Povinné prvky návrhu (projektové studie):

- Základní informace o tvůrcích systému, v případě externí firmy její specifikace, dále pokud jde o systém složený z několika podsystémů také informace o jejich dodavatelích.
- Základní informace o organizaci, pro kterou je systém vyvíjen, včetně uvedení týmu zaměstnanců, kteří popřípadě budou spolupracovat s externí firmou.
- Popis současného stavu organizace.
- Globální návrh IS, nebo - li logický datový model, který je návrhem funkcí a dat systému bez ohledu na technologické prostředí.
- Detailní návrh IS, neboli fyzický datový model, který obsahuje funkční analýzu systému, datovou analýzu, popis veškerých datových toků v organizaci a popis

funkcí řízených událostmi. Celkovým výstupem je návrh funkcí a dat budoucího systému, které jsou definovány na základě prostředí, ve kterém bude systém implementován.

- Detailní popis nasazení IS v praxi, SW a HW studie související s nasazením nového IS.
- Detailní popis testovacího provozu systému, včetně poskytování záručního servisu.
- Celkový harmonogram spolupráce, do něhož patří časový harmonogram dodávky, platby, celková cena, podmínky dodání, ceny pozáručního servisu a podobně.

Celá studie by měla být vytvářena s vědomím, že je to poslední dokument, se kterým se management setká před konečným rozhodnutím o realizaci systému. V případě dohody mezi firmou a tvůrci systému tato studie slouží jako podklad realizace systému a podklad pro podmínky předání a testování.

2.1.4 Implementace

Tato část životního cyklu IS je vlastním programováním, kterého se účastní vybraní programátoři (devolopeři) a analytik nesoucí zodpovědnost za správnost řešení. Podkladem pro jejich práci jsou veškeré informace shromážděné předchozími etapami vývoje IS a fyzický návrh systému.

Transformace návrhu do programů je prováděna v nějakém konkrétním programovacím jazyce jako například (C, C++, C#, Java, Visual Basic atd.).

Postup práce je následující:

Na základě získaných faktů z návrhu se definují vstupy a výstupy jednotlivých operací a určí způsob jejich modifikace. Naprogramují se veškeré funkce a poté se doladí jejich vzájemné propojení. Dále se jednotlivé realizované funkce ověří a připraví se testovací data, která musí obsahovat maximální procento konečných reálných dat.

2.1.5 Testování

V této fázi se provádí připravené testy na naimplementovaném IS. Je nutné vyzkoušet veškeré možné reakce systému na zadávaná data, projít všechny možné větve systému a zjištěné nedostatky opravit. Jeho úkolem je odhalit co nejvíce chyb ve vytvořených programech, ověřit, že implementace vyhovuje specifikaci a splňuje všechny kvalitativní požadavky. Testování se často provádí na systému, který ještě není v reálném prostředí, neboť případné selhání by mohlo mít rozsáhlé následky. Příkladem jsou systémy ve zdravotnictví, letectví, jaderném průmyslu a podobně.

2.1.6 Zavádění systému

Zaváděním systému je míněna především jeho instalace, zavedení do provozu organizace, transformace původní datové základny tak, aby byla přístupná novému systému, poskytnutí manuálů a školení uživatelům. Při školení je nejlepším postupem nejprve školit vedoucí pracovníky a pokračovat zaměstnanci v provozu.

Tato etapa se nesmí v žádném případě podcenit, neboť jejím zanedbáním by mohla u budoucích uživatelů vzniknout averze vůči novému systému a tím neúspěch celého projektu.

Zavedení systému může být provedeno jedním z následujících způsobů:

- **Souběžná strategie** – je založena na pokračujícím provozu původního systému a současném provozu nového systému. Souběžný provoz obou systémů trvá několik pracovních cyklů, dokud není ověřeno, že nový systém pracuje spolehlivě a uživatelé s ním jsou dostatečně seznámeni. Tato metoda je bezpečná, ale náročná pro zaměstnance, jelikož jsou nuceni provádět dvakrát totéž. To by mohlo vést k averzi vůči novému systému, proto se na toto období najímají externí pracovníci.
- **Pilotní strategie** – je založena na zavedení nového systému jen v části podniku a po ověření správné funkčnosti se systém zavede do celé organizace. Jako pilotní část se vybere ta část podniku, která je poměrně náročná a je možné na ni ověřit co nejvíce problémových oblastí.
- **Postupná strategie** – využívá se převážně u velice složitých systémů, kde jsou složité vnitřní vazby. Nejprve se zavádějí primární části IS na kterých ostatní části závisí, po jejich ověření se podobným postupem zavádí ostatní části až po zavedení celého systému.
- **Nárazová strategie** – spočívá v odstranění původního systému a zavedení kompletního nového systému. Tato strategie je velice riskantní, ale ušetří se při ní čas i pracovní síly.

2.1.7 Zkušební provoz

Zkušební provoz je celková realizace projektu, ve které je poskytovatel povinen zajistit okamžitý servis, odstranit chyby zjištěné během provozu nebo dořešit dodatečné požadavky uživatelů v rámci původního návrhu.

2.1.8 Rutinní provoz a údržba

Tato etapa je závěrečnou fází projektu, ve které je systém provozován a používán. Do této etapy také spadá údržba systému, tedy zajištění správného provozu, úprava parametrů aplikací nebo změny některých programů tak, aby splňovaly nové požadavky uživatelů. Mezi základní povinnosti zajištění provozu IS patří organizace prací na počítačích a v síti tak, aby byl zajištěn soulad s původním projektem a dokumentací, zajištění přístupových práv k jednotlivým

aplikacím, sledování činnosti počítačů a síťových prostředků z hlediska výkonu a poruchovosti, zajištění optimálního provozu systému, zabezpečení systému a ochrana dat před neoprávněným přístupem, nebo minimalizace škod vzniklých výpadkem systému (např. záložními systémy nebo archivací dat). V neposlední řadě do této etapy také patří i opětovné školení uživatelů.

2.1.9 Reengineering

Tato etapa je přehodnocením požadavků na systém, a pokud je nelze již splnit pouhou úpravou, je krokem vedoucím na počátek životního cyklu. [1], [2]

2.2 Modely životního cyklu systému

Některý software se vyvíjí s přísnou disciplínou, některý vývoj je mírně řízen a jiný software je vytvořen bez jakéhokoliv řízení. Způsob, který je při tvorbě softwaru použit se odrazí na kvalitě výsledného produktu. Zákazník obvykle nakonec pozná, jaký postup, byl při tvorbě softwaru použit.

Proces, podle kterého se vytváří softwarový produkt, a to od jeho prvotního záměru do veřejného uvedení na trh, se nazývá model životního cyklu vývoje softwaru.

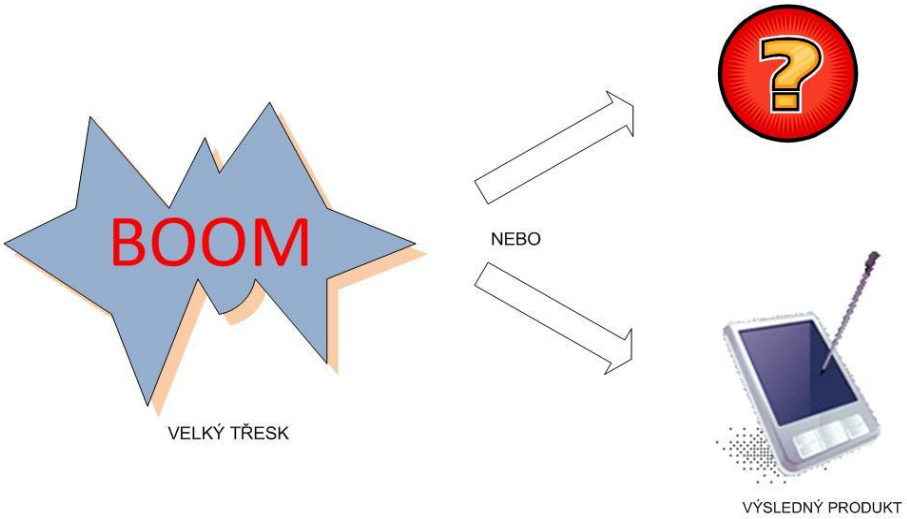
Při vývoji softwaru je možné použít celou řadu různých metod, přičemž žádný model se nedá ani u konkrétního projektu označit za jediný správný. Většina metod je pouhými variantami následujících čtyř nejčastěji používaných modelů:

- Model velkého třesku.
- Model “programuj a opravuj”.
- Model vodopádu.
- Spirálový model.

Každý z těchto modelů má své výhody i nevýhody. Konkrétnímu typu modelu, použitému v projektu je nutné přizpůsobit postup při testování softwaru.

2.2.2 Model velkého třesku

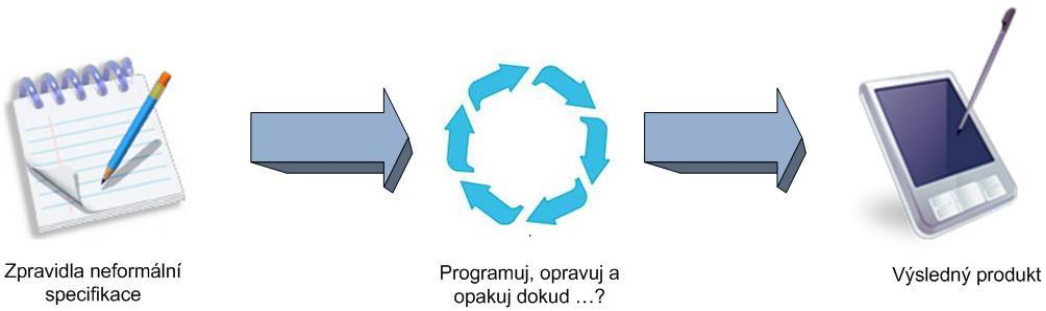
Princip tohoto modelu je jednoduchý, na jedné straně stojí lidé a zdroj peněz a na druhé straně množství vydané energie, z čehož vznikne softwarový produkt. Tento princip je zobrazen na obr. X. Princip modelu velkého třesku je velice jednoduchý. Je k němu zapotřebí jen velmi málo plánování, rozvrhování práce a formálního vývoje, pokud vůbec nějaké plánování obsahuje. Veškeré úsilí se soustřeďuje jen na samostatný vývoj softwaru a psaní jeho programového kódu. Pokud nejsou vstupní požadavky na produkt příliš srozumitelné a datum odevzdání produktu je velice volné, je tento proces ideální. Nikdo ovšem předem neví, jak bude výsledný program přesně vypadat.



Obrázek 2.1 Model velkého třesku

2.2.3 Model programuj a opravuj

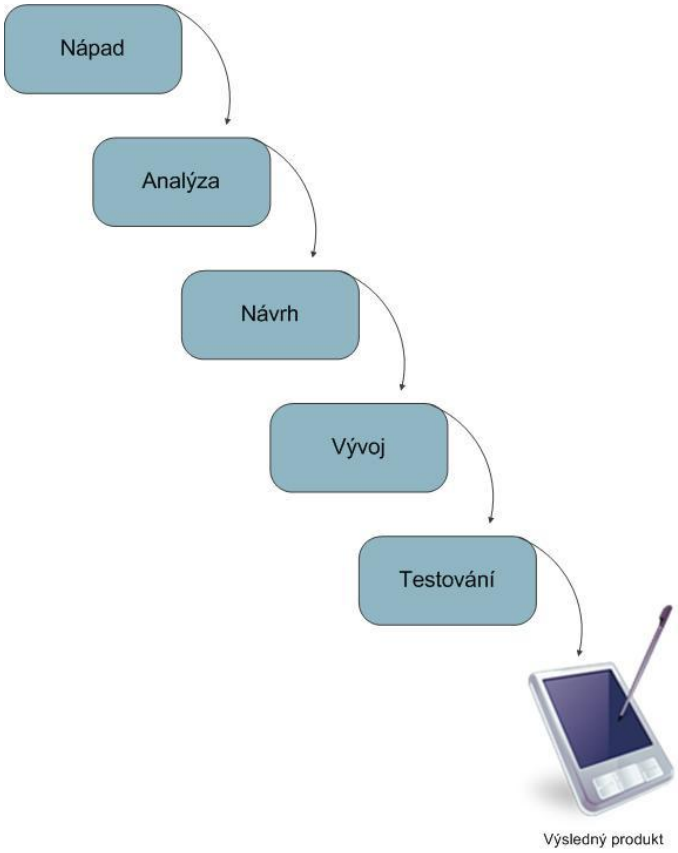
Tento model představuje oproti modelu velkého třesku krok vpřed, protože alespoň vyžaduje určitou představu o požadavcích na produkt.
Na počátku tohoto modelu je hrubá představa výsledného produktu. Je sestaven jednoduchý návrh a poté následuje dlouhý, neustále se opakující cyklus kódování (programování), testování a opravování chyb. V jistém okamžiku se později rozhodne, že je na čase skončit a pustit výsledný produkt do světa.



Obrázek 2.2 Princip modelu programuj a opravuj

2.2.4 Model vodopádu

Model vodopádu bývá obvykle prvním, který se při výuce programování přednáší. Je jednoduchý, elegantní, dává smysl a u správných projektů velice dobře funguje. Jednotlivé kroky tohoto modelu popisuje obrázek 2.3. V tomto modelu proces vývoje softwaru “přetéká” z jednoho kroku do dalšího.



Obrázek 2.3 Princip modelu vodopádu

Projekt vyvíjený v modelu vodopádu postupuje jakoby “dolů” v posloupnosti kroků, které začínají u prvotní myšlenky (nápadu) a končí u výsledného produktu. Na konci každého z kroků vývojový tým zhodnotí, zda se může pustit, do dalšího kroku. Pokud ještě projekt není zralý k postupu do dalšího kroku, zůstává ve stejné úrovni, dokud tato úroveň není dokončena.

U modelu vodopádu jsou tři důležité poznatky:

Velký důraz je kladen na specifikaci výsledné podoby produktu. Na obrázku je vidět, že fáze vývoje, nebo-li kódování, se nachází v jediném bloku.

Jednotlivé kroky jsou diskrétní a vůbec se nepřekrývají.

Ve vodopádu není cesty zpět. Jakmile se nacházíte v určité úrovni, musíte dokončit všechny úkoly, které na vás v tomto kroku čekají, a poté musíte přejít dále – vracet se již nemůžete. (Pozn. U různých variant modelu vodopádu jsou tato pravidla poněkud oslabena, takže jednotlivé kroky se mohou částečně překrývat a v případě potřeby je možné se vrátit o jeden krok zpět).

Tento postup se může na první pohled zdát velice omezující, ale u projektů s dobře srozumitelnou definicí produktu a disciplinovaným vývojovým týmem dává velice dobré

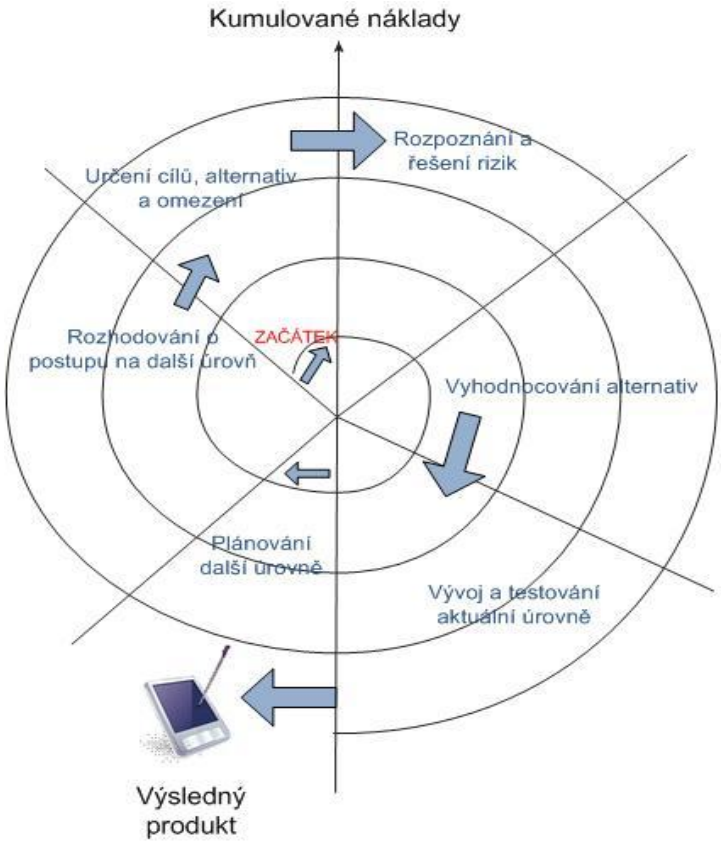
výsledky. Cílem je ještě před napsáním prvního řádku programového kódu rozpracovat všechny neznámé a popsat všechny detaily. Nevýhodou takovéhoto striktního postupu ovšem je, že zejména v dnešní překotné době, kdy se produkty vyvíjejí na Internetu, již v okamžiku pečlivého promyšlení a nadefinování všech aspektů softwarového produktu se prvotní důvod pro jeho vývoj může zcela změnit.

2.2.5 Spirálový model

Spirálový model výrazným způsobem řeší problémy ostatních modelů a navíc má své vlastní příjemné stránky.

Spirálový model zavedl Barry Boehm v roce 1986 ve svém dokumentu asociace ACM (Association for Computin Machinery), nazvaném „A Spiral Model of Software Development and Enhancement“ (Spirálový model vývoje a zdokonalování systému). Používá se poměrně často a ukázalo se, že je to pro vývoj softwaru opravdu velice efektivní postup.

Základní myšlenku, na které je spirálový model postaven, je možné popsat následovně: Úplně na začátku nemůžeme definovat vše do úplných podrobností. Začneme tedy s malou definicí, těch nejdůležitějších funkcí, vyzkoušíme si je, vyžádáme si připomínky od zákazníků a poté přejdeme na další úroveň vývoje. Celý proces opakujeme, dokud se nedostaneme k výslednému produktu.



Obrázek 2.4 Princip spirálového modelu

Při každém průchodu spirálou provádíme následujících šest kroků:

- určení cílů, alternativ a omezení;
- rozpoznání a řešení rizik;
- vyhodnocení alternativ;
- vývoj a testování aktuální úrovně;
- plánování další úrovně;
- rozhodnutí o postupu na další úroveň.

Součástí spirálového modelu je částečně model vodopádu (také zde máme kroky analýzy, návrhu, vývoje a testování), a částečně také model "programuj a opravuj" (v každém průchodu spirálou) a dokonce zde vidíme kousek velkého třesku (při pohledu zvnějšku). Když se k tomu přidá snížení nákladů na chyby díky včasnému odhalení problémů, vznikne velice dobrý model vývoje. [3]

2.3 Modely životního cyklu z hlediska testování

Jelikož se tato diplomová práce zabývá kromě analýzy a návrhu, také testováním, je potřeba se zamyslet nad tím, jaký vliv na testování budou mít modely použité při vývoji systému.

2.3.2 Model velkého třesku z hlediska testování

Ve většině případů se při modelu velkého třesku žádné, nebo téměř žádné formální testování neprovádí, což je vidět i na obrázku 2.1. Jestliže se vývojový tým k nějakému testování přece jen uchýlí, odehraje se zpravidla těsně před konečným uvedením produktu na trh. Je záhadou, proč se v takových případech do modelu velkého třesku testování dostane, snad je to způsobeno tím, že mají členové týmu dobrý pocit, že bylo alespoň něco testováno.

Dostanete – li za úkol testovat softwarový produkt, vyvíjený v modelu velkého třesku, znamená to snadnou a zároveň obtížnou práci. Software je již téměř hotový, takže máte dokonalou specifikaci (tou je samotný produkt). A protože je prakticky nemožné se v projektu vracet a opravovat věci, které nefungují, nezbývá vám nic jiného, než nalezené chyby sepsat, aby zákazníci alespoň věděli, do čeho jdou.

Nevýhodou takovéhoto testování je, že z pohledu řízení projektu je již produkt připraven k expedici, takže vaše práce v podstatě jen zdržuje dodání zákazníkovi. Čím déle budete na testování pracovat a čím více chyb v produktu najdete, tím více se dostáváte pod dvojí tlak.

2.3.3 Model „programuj a opravuj" z hlediska testování

V modelu „programuj a opravuj" se žádné velké testování výslovně neprovádí, podobně jako u modelu velkého třesku. Přesto zde ale představuje významný spojovací článek mezi kódováním a opravováním.

Tester projektu vyvíjeného v modelu „programuj a opravuj“ si musí uvědomit, že bude spolu s programátory pracovat v neustálém cyklu. Prakticky každý den dostane na stůl novou nebo alespoň aktualizovanou verzi softwaru a bude ji muset vyzkoušet. Provede tedy potřebné testy, oznámí nalezené chyby a hned dostane další verzi softwaru. Často se ani nepodaří pořádně dokončit testování jedné verze a už má před sebou další, která má třeba některé funkce změněné nebo zcela nové. K otestování většiny věcí se nakonec dostane, v softwaru bude nacházet stále méně chyb a úplně nakonec se někdo rozhodne, že je na čase produkt vypustit na trh.

S tímto modelem se tester softwaru setká při své práci zřejmě nejčastěji. Je to dobrý úvod do vývoje softwaru, díky němuž tester lépe ocení vyšší, formálnější metody vývoje.

2.3.4 Model vodopádu z hlediska testování

Model vodopádu má od předchozích metod tu výhodu, že je vše předem pečlivě popsáno a specifikováno. Software se předává testerům v okamžiku, kdy již je o každém jeho detailu rozhodnuto a vše je implementováno do softwaru. Testovací skupina pak přesně ví, co testuje a nikdy není pochyb o tom, zda je určité chování systému chyba či ne. Díky tomu je možné vytvořit jasný plán a rozvrh testů pro tento systém.

Tento model má také jednu nevýhodu, testování se zde totiž provádí vždy až na konci vývojového cyklu. Důsledkem pak může být, že na začátku vývoje dojde k problému, na který se přijde až v době, kdy již má jít produkt podle rozvrhu prací na trh, v tu dobu již jsou náklady na odstranění chyb velice vysoké, ne – li neúnosné.

2.3.5 Spirálový model z hlediska testování

U spirálového modelu se testování provádí již v počátcích vývoje systému a tester je zapojen i do předběžných fází vývoje. Tester tudíž ví, kam vývoj softwaru směřuje. Dostává nové verze softwaru k testování průběžně (v každém průběhu spirálou) a tím tedy může zabránit nalezení závažných chyb až v pozdní fázi vývoje, kde je již náprava chyb velice drahá, navíc není tlačen až na konci aby bylo veškeré testování provedeno najednou a na poslední chvíli. Jelikož tester testuje software neustále, v posledním průchodu již jen ověří, zda opravdu vše funguje tak jak má. [3]

3 Systémové požadavky

Požadavky udávají, jak bude výsledný systém pracovat, jaké bude mít funkce, omezení nebo také jak by měl vypadat vzhled uživatelského rozhraní. Nejsou – li veškeré požadavky přesně specifikovány, může dojít k nedorozumění mezi zadavatelem a vývojářem, kdy zadavatel předpokládá nějakou pro něj základní funkci, která vývojáři až tak zřejmá není. Poté může být pro obě strany velice nepříjemné, zjistit u výsledného systému, že něco nepracuje tak jak má, nebo tam třeba některá z funkcí není a to jen proto, že nebyla udána jako požadavek na systém. Veškeré požadavky musí být přehledně sepsány, třeba i proto, aby při případné výměně vývojáře nemusely být požadavky specifikovány znovu. Dokument, do kterého jsou požadavky následně sepsány, se nazývá „systémová specifikace“.

Ve slovníku softwarové technologie z roku 1990 je požadavek definován jako:

- Podmínka nebo funkce, kterou uživatel potřebuje pro řešení problému nebo dosažení nějakého cíle.
- Podmínka nebo funkce, kterou musí systém nebo jeho část splňovat, aby vyhověl smlouvě, standardu, specifikaci nebo jinému dokumentu, jenž se na něj formálně vztahuje.
- Dokumentovaná podoba některého z předchozích dvou bodů.

Požadavky dělíme na:

- **Podnikatelské**
  Formulují cíle organizace či zákazníka, který zadal požadavek na vývoj daného systému. Tyto požadavky říkají, z jakého důvodu firma či organizace požaduje daný systém.
- **Uživatelské**
  Tyto požadavky udávají, jaké úkoly nebo jaké operace se systémem bude uživatel schopen provádět. Tyto požadavky mohou být zapsány formou případů užití a jejich scénářů, také tabulkami obsahujícími reakce na různé situace.
- **Funkční**
  Udávají přesně, co musí programátoři naprogramovat, aby byl uživatel plnit své úkoly. Mohou být také nazývány požadavky na chování.

Vlastnosti správně zapsaných požadavků

Jednoznačnost – požadavky mají být sepsány formou jednoduchých a stručných vět, tak aby výklad nenabýval více významů. Uživatel taky požadavkům a tomu jak jsou sepsány musí rozumět.

Nepostradatelnost – každý požadavek, by měl být funkce, kterou si vyžádal zákazník, nebo jej požaduje norma atd. Zdrojem požadavků by měl být vždy někdo, kdo má právo požadavek zadat.

Ověřitelnost – schopnost ověřit správnost implementace požadavku nějakým testem, zkouškou atd. Nelze – li požadavek ověřit, způsob, jak by se měl implementovat je potom věcí názoru. Mezi neověřitelné požadavky patří požadavky neproveditelné, nejednotné, neúplné a nejednoznačné.

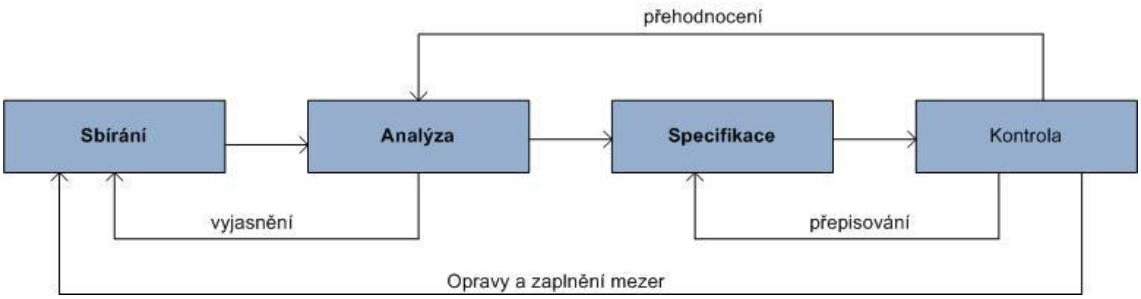
Priorita – každý požadavek, má mít přiřazenu prioritu, říkající, jak je daná funkce důležitá pro vydání systému.

Proveditelnost – každý požadavek musí být v rámci známých možností a omezení na systém proveditelný.

Správnost – požadavek musí danou funkcionalitu popisovat přesně, je – li například požadavek v rozporu se svým nadřazeným požadavkem, není to správně. Zda je požadavek správný může ověřit pouze uživatel.

Úplnost – požadavek musí obsahovat všechny potřebné informace o dané funkcionalitě potřebné k její implementaci, tzn. musí být úplný.

Požadavky na vyvíjený systém se v průběhu seznamování se s danou problematikou a její analýzy vyvíjejí. Tento děj probíhá iterativně a to ve čtyřech fázích - sbírání, analýza, specifikace a kontrola viz. obrázek 3.1. Sbírání požadavků zahrnuje poslouchání uživatelů, komunikace s nimi, pozorování jejich práce a vyptávání se. V analýze jsou požadavky zpracovávány, tříděny do kategorií a dle potřeb zákazníka přepsány na funkční požadavky. Ve specifikaci jsou veškeré informace od zákazníka a požadavky sepsány do různých typů dokumentů a diagramů a při kontrole má uživatel za úkol ověřit, zda je specifikace přesná a úplná a jsou opraveny případné chyby. Tento proces se během vývoje požadavků několikrát opakuje. [4]



Obrázek 3.1 Vývoj požadavků

4 Systémová specifikace

Specifikace požadavků je jedním ze způsobů, jak zapsat softwarové požadavky. Jiným způsobem je strukturovaný a pečlivě psaný dokument v přirozeném jazyce, nebo použití grafických modelů, což je nejpraktičtější nástroj pro dokumentaci požadavků. Systémová specifikace patří mezi nejpřísnější a nejpřesnější způsoby zapsání softwarových požadavků.

4.1 Co je systémová specifikace

Systémová specifikace může být také nazývána *specifikace požadavků*, *funkční specifikace*, *produktová specifikace nebo požadavková specifikace.*
Specifikace požadavků slouží k popisu funkcí, které má vyvíjený systém obsahovat a taky různá omezení, která musí splňovat. Popisuje chování systému za různých podmínek, které mohou nastat (chování je popisováno tak, aby zahrnovalo veškeré potřebné detaily). Naopak návrh, stavba, testování a podrobnosti řízení projektu by specifikace obsahovat neměla. Specifikace slouží jako výchozí dokument pro následné plánování, návrh, implementaci a poté i pro testování a psaní uživatelské dokumentace. Z toho také vyplývá, pro koho je specifikace určena.

Seznam možných čtenářů systémové specifikace:

- Projektoví manažeři
  - podle specifikace plánují odhady termínů, náročnost a zdroje.
- Vývojářský tým
  - podle specifikace provádí implementaci systému.
- Testovací tým
  - používá specifikaci k plánování testů (vývoj testovacích plánů, scénářů a postupů testování).
- Technická podpora, údržbáři
  - Specifikace jim slouží jako dokument, podle kterého mohou zjistit, jak by se měla některá část systému chovat.
- Dokumentátoři
  - Specifikace slouží jako výchozí dokument pro psaní uživatelské dokumentace, manuálů a nápověd.
- Školitelé
  - Navrhují podle specifikace a uživatelské dokumentace materiály pro výuku.
- Právní oddělení
  - Podle specifikace zjistí, zda systém vyhovuje zákonům.
- Subdodavatelé
  - Pracují podle systémové specifikace.
- Zákazníci, marketingové oddělení, obchodní oddělení

- Specifikace jim prozradí, jak bude vypadat výsledný produkt a co od něj mohou očekávat.

Systémová specifikace musí být podrobná. Systém bude mít takové funkce, které jsou popsány ve specifikaci, proto, není – li nějaká funkce ve specifikaci popsána, nelze očekávat, že bude implementována v systému.

4.2 Šablona pro psaní specifikace a její implementace na navrhovaný systém

Šablony pro specifikaci by měly mít všechny firmy, zabývající se vývojem softwaru. Jsou to standardní šablony pro specifikaci firemních projektů. Mezi standardní šablony patří například Davis 1993, Robertsonová a Robertson 1999, Lefingwell a Widrig 2000. Často používanou šablonou jsou šablony odvozené od standardu IEEE 830-1998.
Pro tuto práci je použita šablona ze standardu IEEE 830. Tato šablona je vhodná pro mnoho druhů projektů, ale má i několik nepřehledných částí. [4]

Šablona pro psaní specifikace vypadá následovně:

1 Úvod

Úvod obsahuje přehled specifikace, aby se v ní čtenář zorientoval a uměl ji použít.

1.1 Předmět specifikace

Popište systém nebo aplikaci, jejímiž požadavky se specifikace zabývá. Nezapomeňte uvést číslo verze. Pokud se specifikace týká jen části celého systému, jasně tuto část vymezte.

Ukázka ze specifikace vyvíjeného systému:

Tato specifikace popisuje požadavky na systém navrhovaný pro pečovatelskou službu, poskytující své služby jak přímo v sídle firmy, tak v domácnosti klientů. Účelem tohoto systému je převod stávajícího systému záznamů pacientů z klasické formy do elektronické podoby. Systém bude kromě elektronického uchovávání záznamů pacientů umožňovat také plánování úkolů zaměstnancům zaměstnavatelem pro daný den, přičemž si své úkoly zaměstnanec zobrazí ve svém pracovním mobilním zařízení.

1.2 Typografické konvence

Popište textové styly, způsob vyznačování nebo důležitější notace. Uveďte například, jestli se priorita obecných požadavků vztahuje i na všechny odvozené podrobné požadavky, nebo jestli má mít každý funkční požadavek svou vlastní prioritu.

Ukázka ze specifikace vyvíjeného systému:

Všechny prvky aplikace budou pojmenovány následovně:
zkratka názvu prvku_Název okna ve kterém se prvek nachází+Název účelu prvku

název typu prvku vždy začíná malým písmem a za ním píšeme vždy podtržítko. Poté následuje zbytek názvu prvku, kde každé slovo začíná velkým písmem (Název prvku je psán strukturou psaní názvů, které se říká Camel).

například: máme – li v okně *Pacient* prvek typu *label* a je to prvek popisující Jméno pacienta, bude se tento prvek nazývat lbl_PatientsPatientName.

1.3 Cílové publikum, návod ke čtení

Napište seznam různých čtenářů, pro které je specifikace určena. Popište, co obsahuje zbytek specifikace, a jak je rozdělený. Doporučte každému z typů čtenářů ideální pořadí pro čtení specifikace.

Ukázka ze specifikace vyvíjeného systému:

Tento dokument je určen ke čtení analytikům, vývojářům aplikace, testerům a zadavateli (zákazníkovi).
Doporučené čtení specifikace je následující:

- Analytik a vývojář – čtou celou specifikaci.
- Tester – čte specifikaci podle typu testů.
- Zadavatel (zákazník) – čte obecný popis specifikace (část 2) a funkce systému (část 3).

1.4 Rozsah projektu

Stručně popište specifikovaný systém a jeho účel. Popište vztah systému k uživatelům, firemním cílům, podnikatelským pravidlům a strategiím. Pokud máte vizi a rozsah projektu popsané samostatným dokumentem, nesnažte se jeho obsah zopakovat a raději se na něj odkažte. Pokud jde o specifikaci další verze inkrementálně vyvíjeného systému, popište také konkrétní rozsah této verze (měl by být podmnožinou dlouhodobé strategické vize produktu).

Ukázka ze specifikace vyvíjeného systému:

Účelem tohoto projektu je modernizace vedení záznamů pečovatelské služby z nynějšího papírového systému do přehlednější elektronické podoby. Data mají být uchovávána v databázi (database machine). Aplikace bude umožňovat správu zaměstnanců agentury, zákazníků - pacientů, plánování pracovních úkolů jednotlivým zaměstnancům prostřednictvím desktopového klienta, přičemž zaměstnanci si mohou zobrazit informace o svých denních úkolech na svých pracovních mobilních zařízeních (PDA). Aplikace bude mít implementovánu

možnost generování statistik a testů pacientů z uchovaných dat. Dále aplikace umožňuje správu spotřebovaného materiálu.

1.5 Odkazy

Napište seznam všech dokumentů a dalších zdrojů, na které se tato specifikace odkazuje, pokud možno i s hypertextovými odkazy. Jde například o různé smlouvy, specifikace systémových požadavků, případy užití, specifikace rozhraní, specifikace souvisejících systémů, standardy pro uživatelská rozhraní a další standardy. Uveďte dostatek informací, aby se čtenář ke každému z odkazovaných dokumentů dostal – například název, autora, číslo verze, datum vydání a zdroj nebo adresu (síťový disk, URL a podobně).

Ukázka ze specifikace vyvíjeného systému:

Dokumentace databáze, systémová specifikace mobilního klienta, dokumentace databáze mobilního klienta.

2 Obecný popis

Tato část specifikace obsahuje obecný přehled systému, provozního prostředí, očekávaných uživatelů a známých omezení, předpokladů a závislostí.

2.1 Kontext systému

Popište původ systému a jeho kontext. Jde o dalšího člena rostoucí rodiny produktů, další verzi stabilního systému, náhradu existující aplikace, úplně nový produkt? Pokud specifikace popisuje komponentu většího systému, jak tato komponenta zapadá do celkového systému a jaká jsou mezi nimi hlavní rozhraní?

Ukázka ze specifikace vyvíjeného systému:

Jedná se o novou aplikaci, bez jakýchkoliv předchozích verzí. Je to aplikace skládající se ze čtyř komponent (celků) a to z „Serverová část systému“, „PC klient pro správu uživatelů a plánování“, „Synchronizační klient“ a z části „Mobilní klient“. Produkt je desktopová a mobilní aplikace s databázovým serverem uchovávajícím data.

2.2 Funkce systému

Popište hlavní funkce systému. Na podrobnosti je vyhrazena celá třetí část specifikace, takže na tomto místě stačí uvést jen obecný přehled. Dobrý by mohl být obrázek hlavních skupin požadavků a vztahů mezi nimi, například obecný diagram datových toků, diagram případů užití nebo diagram tříd.

Ukázka ze specifikace vyvíjeného systému:

Funkce s daty

- práce s uživatelem
- práce s doktorem
- práce s pacientem
- práce s plány
- práce s historií
- práce s testy a statistikami

2.3 Třídy uživatelů

Popište třídy uživatelů, kteří podle vás budou produkt používat, a jejich hlavní vlastnosti. Některé požadavky se mohou týkat jen některých tříd uživatelů. Zjistěte, kterým třídám uživatelů je potřeba dávat přednost. Třídy uživatelů patří mezi účastníky projektu, kteří jsou podrobně popsáni v dokumentaci vize a rozsahu projektu.

Ukázka ze specifikace vyvíjeného systému:

Hl. sestra – tento uživatel je uživatel nadřízený všem zdravotním pracovníkům v agentuře. Uživatel je oprávněn provádět správu uživatelů (vytvoření, editaci, odstranění) a zákazníků agentury. Dále je oprávněn přidělovat úkoly zaměstnancům (plánovat procedury zákazníkům) a kontrolu jejich provedení. Uživatel hlavní sestra má také oprávnění spravovat číselníky, jejichž funkce byla popsána již u vedoucího systému.

2.4 Provozní prostředí

Popište prostředí, ve kterém systém poběží – včetně hardwaru, typu a verze operačního systému a zeměpisného umístění uživatelů, serverů a databází. Napište seznam všech softwarových komponent nebo aplikací, se kterými musí systém bez problémů vycházet. Některé z těchto informací může v obecné rovině popisovat již dokumentace vize a rozsahu projektu.

Ukázka ze specifikace vyvíjeného systému:

Serverová část systému – tato část poběží v kancelářském prostředí klienta používajícího produkt. K realizaci bude použito klasické stolní PC vybavené následujícími softwarovými prostředky – database machine(databázové prostředí – např. MS SQL Server express edition), .NET compact framework 3.5, IIS (internetový informační server), OS (operační systém Windows XP/Vista, windows server).

2.5 Omezení návrhu a implementace

Popište libovolné faktory, které omezují možnosti vývojářů, a důvody těchto omezení. Mezi běžná omezení patří například:

- Konkrétní technologie, nástroje, programovací jazyky nebo databáze, které vývojáři musí nebo naopak nesmí použít.
- Omezení daná provozním prostředím systému, například typ a verze použitého webového prohlížeče.
- Požadované vývojářské konvence nebo standardy. (Například pokud si chce zákazník software udržovat sám, vaše organizace může vybrat konkrétní notaci pro zápis návrhu a kódu, kterých se pak musí držet i všichni subdodavatelé).
- Zpětná kompatibilita s předchozími produkty.
- Omezení daná podnikatelskými pravidly (která se dokumentují jinde).
- Stávající standardy pro uživatelské rozhraní, které se musí při vylepšování existujícího systému dodržovat.
- Standardní formáty pro výměnu dat, například XML.

Ukázka ze specifikace vyvíjeného systému:

Aplikace bude napsána s použitím technologie .NET verze 3.5, programovací jazyk bude použit libovolný kompatibilní s .NETem. Databázová vrstva musí být vytvořena v prostředí MS SQL server, přičemž k neobjektovým datům bude přistupováno pomocí technologie LINQ. Uživatelské rozhraní bude vytvořeno buď pomocí frameworků Visual Studia, nebo jako WPF. Veškerá zařízení, na kterých poběží aplikace, musí obsahovat .NET framework 3.5 (mobilní zařízení .NET Compact framework 3.5). Mobilní zařízení musí také obsahovat Wi-Fi. Funkčnost aplikací budou zajišťovat webové služby.

2.6 Uživatelská dokumentace

Napište seznam všech součástí uživatelské dokumentace, která se bude se systémem dodávat. Mezi uživatelskou dokumentaci patří například uživatelské manuály, online nápověda a průvodci. Zjistěte, v jakých formátech se dokumentace musí dodávat, které standardy musíte dodržet a jaké nástroje použít.

Ukázka ze specifikace vyvíjeného systému:

Veškerá dokumentace bude dodána ve formátu PDF, psána bude dle vytvořené šablony (viz. příloha).

Součástí dokumentace bude:

- Uživatelský manuál pro PC klienta
- Uživatelský manuál pro mobilního klienta
- Nápověda v aplikaci PC klienta
- Nápověda v aplikaci mobilního klienta

Dokumentace nebude obsahovat online nápovědy ani průvodce aplikací.

2.7 Předpoklady a závislosti

Předpoklad je tvrzení, které se automaticky považuje za pravdivé. Pokud předpoklad pravdivý není, nevědí o něm všichni nebo se změní, dojde k problémům, takže některé předpoklady jsou pro projekt rizikové. Jeden čtenář specifikace může předpokládat, že systém bude odpovídat nějakému standardu pro uživatelská rozhraní, a další čtenář si uživatelské rozhraní představí jinak. Vývojář může předpokládat, že se určitá skupina funkcí napíše přímo pro danou aplikaci, zatímco analytik předpokládá, že se recykluje kód z předchozího projektu a projektový manažer najisto počítá s pořízením nějaké komerční knihovny.

V této části specifikace také popište všechny závislosti projektu na vnějších okolnostech mimo jeho vliv, například plánovaném termínu vydání příští verze operačního systému nebo vydání nějakého průmyslového standardu. Pokud chcete do systému integrovat nějaké komponenty, které se vyvíjí v rámci jiného projektu, váš projekt závisí na úspěšném dokončení těchto cizích projektů ve stanoveném termínu. Pokud už jsou závislosti popsané jinde, například v projektovém plánu, jen se na příslušný dokument odkažte.

Ukázka ze specifikace vyvíjeného systému:

Každá funkce aplikace bude zpřístupněna pouze uživateli, který má možnost tyto funkce vykonávat (uživatel musí být přiřazen k roli, které je funkce, kterou chce provádět přiřazena - povolena). Funkce, které nemá daný uživatel povoleny, mu budou znepřístupněny ihned po přihlášení uživatele do systému.
Vývoj systému není závislý na žádných neovlivnitelných vnějších okolnostech.

3 Funkce systému

Tato šablona je uspořádána podle funkcí systému, což je jen jedna z možností, jak lze funkční požadavky setřídit. Dalšími možnostmi je uspořádání podle případů užití, pracovních režimů, uživatelských tříd, událostí, odpovědí, objektových tříd nebo funkční hierarchie (IEEE 1998b). Jednotlivé způsoby setřídění je možno také kombinovat (můžete tak získat například specifikaci, která je uspořádána podle objektových tříd, v nichž je text dělen dle událostí). Analytik si vybere takové rozdělení, ze kterého čtenář snadno pochopí, jaké schopnosti jsou u vyvíjeného systému plánovány.

3.x Funkce x

Název funkce by měl být jednoduchý (shrnutí funkce systému v několika slovech), snadno pochopitelný a věcný. (například: 3.1 Přihlášení uživatele). Podčásti 3.x.1 … jsou pro všechny funkce systému stejné.

3.x.1 Popis a priorita

Obsahuje stručný popis funkce a její zařazení do skupiny podle její priority (vysoká, střední, nízká). Priority se v průběhu projektu mění – jsou dynamické. Je – li použit nějaký nástroj pro správu požadavků, je třeba nadefinovat prioritu jako požadavkový atribut.

3.x.2 Události a odpovědi

Tato část specifikace obsahuje seznam událostí (akcí uživatele nebo signálů příchozích z vnějších zařízení) a následné chování systému odpovídající těmto událostem.

3.x.3 Funkční požadavky

Zde jsou podrobně popsány veškeré funkční požadavky odpovídající dané funkci systému. Funkční požadavek udává, co vše musí systém umět, aby byl schopen vykonat danou funkci. Nesmí také chybět reakce systému na očekávané chybové stavy nebo chybné akce a vstupy. Každý funkční požadavek musí být označen jedinečným identifikátorem.

4 Požadavky na vnější rozhraní

V této části specifikace popisuje hardwarové, softwarové nebo databázové systémy, se kterými musí systém komunikovat. Zajišťuje, aby komunikace systému s externími komponentami probíhala bez problémů. Mají – li různé části systému různá vnější rozhraní, měla by být tato část specifikace uvedena mezi podrobnými požadavky na každou z těchto částí.

Podrobný popis datových a řídicích částí rozhraní náleží do datového slovníku, složitější systémy s větším počtem komponent by měly mít svou vlastní specifikaci rozhraní, nebo specifikaci systémové architektury. Potřebuje – li dokumentace rozhraní materiály z ostatních dokumentů, měla by se na ně odkazovat (např. na specifikaci programátorského rozhraní, manuál hardwarového zařízení se seznamem chybových kódů, které může zařízení vrátit).

4.1 Uživatelská rozhraní

Popisuje logické vlastnosti všech uživatelských rozhraní systému. Mezi důležité vlastnosti patří:

- Odkazy na GUI standardy nebo grafické manuály příslušné rodiny produktů.
- Standardy týkající se fontů, ikon, popisků, obrázků, barevné palety, pořadí prvků, běžně používaných ovládacích prvků a podobně.
- Omezení týkající se rozlišení obrazovky nebo rozložení prvků na obrazovce.
- Standardní tlačítka, funkce nebo navigační odkazy, které musí obsahovat každá obrazovka (např. tlačítko nápovědy).
- Klávesové zkratky.
- Pravidla pro zobrazení zpráv.
- Pravidla usnadňující lokalizaci softwaru.
- Pomůcky pro zrakově postižené uživatele.

Podrobný návrh rozhraní (např. konkrétní podoba dialogů) se popisuje v samostatné specifikaci uživatelského rozhraní, ne ve specifikaci požadavků. Naopak návrhy obrazovek mohou být vloženy do specifikace požadavků.

Ukázka ze specifikace vyvíjeného systému:

Uživatelské rozhraní bude pouze jedno, ovšem pro uživatele s jiným oprávněním než administrátorským budou skryty funkce, ke kterým nebude mít přihlášený uživatel přístup.

- GUI bude obsahovat standardní ovládací prvky definované .NET frameworkem 3.5.
- Každá obrazovka musí nutně obsahovat tlačítko pro vrácení do seznamu aktuálního okna (například seznamu pacientů, jsem – li v okně pacientů) a tlačítko pro odhlášení uživatele.
- Každá zobrazená ***chybová zpráva*** musí obsahovat stručný popis chyby, která nastala (např. uživatelské jméno nebo heslo je neplatné).

4.2 Hardwarové rozhraní

Popisuje vlastnosti každého rozhraní mezi softwarovými a hardwarovými komponentami systému (např. seznam podporovaných typů zařízení, datová a řídicí rozhraní mezi softwarem a hardwarem, komunikační protokoly).

Ukázka ze specifikace vyvíjeného systému:

Systém nepočítá s žádnými hardwarovými rozhraní.

4.3 Softwarová rozhraní

Zde specifikace popisuje propojení vyvíjeného systému s konkrétními verzemi jiných softwarových komponent, jako jsou například:

- operační systémy,
- knihovny,
- nástroje,
- databáze,
- integrované komerční komponenty.

Je zde popsán účel řídicích signálů, zpráv či dat, které si systém a softwarové komponenty mezi sebou vyměňují, služby na kterých jsou závislé některé externí softwarové komponenty a princip komunikace mezi jednotlivými komponentami. Dále jsou zde označena data, která jsou komponentami sdílena. Potřeba implementace sdílení dat určitým daným způsobem (pomocí sdílené datové oblasti) je uvedena v části specifikace *„2.5 omezení návrhu a implementace“*.

Ukázka ze specifikace vyvíjeného systému:

Systém bude propojen s databázovým serverem, tento server bude spravovat, uchovávat a poskytovat veškerá data, se kterými systém pracuje (viz. dokumentace databáze). Systém bude s databází propojen buď síťovým kabelem, nebo budou se systémem uloženy na stejném PC. Systém bude k datům přistupovat pomocí technologie LINQ. Databázový server poběží na operačním systému Windows XP/Vista a to z důvodu, že chceme použít pro vývoj databáze MS SQL server a pro vývoj systému .NET framework 3.5.

4.4 Komunikační rozhraní

V této části jsou popsány požadavky na komunikační funkce, které bude systém používat (např. webový prohlížeč, elektronické formuláře, protokoly pro síťovou komunikaci, e-mail), formátování vyměňovaných zpráv a zabezpečení komunikačních kanálů (šifrování zpráv, synchronizační mechanismy, přenosové rychlosti).

Ukázka ze specifikace vyvíjeného systému:

Pro komunikaci mobilního klienta s databází bude použit standard pro lokální bezdrátové sítě Wi-Fi, využíván bude protokol TCP/IP.

5 Další parametrické požadavky

Tato část popisuje parametrické požadavky s výjimkou požadavků na vnější rozhraní opsaných v části 4 a omezení, popsaných v části 2.5.

5.1 Výkonnostní požadavky

Uvádí požadavky na výkon jednotlivých funkcí systému, a také důvody pro tyto výkonnostní požadavky. Pro jednodušší rozhodování vývojáře je dobré uvést požadovaný počet transakcí za sekundu, časy odezvy, výpočetní přesnost, u real – time systémů je také třeba uvést vzájemné vztahy mezi jednotlivými časy. Také je možno uvést nároky na zatížení větším počtem uživatelů, maximální počet řádků v databázových tabulkách, paměť nebo disk.

Ukázka ze specifikace vyvíjeného systému:

Aplikace nebude pracovat v real – time režimu, data budou dodávána do aplikace v kolekcích, to znamená, že nebude třeba stále přistupovat k databázi, ale data se stáhnou pouze jedenkrát a zapíší do databáze až při uložení. Aplikace tedy nebude náročná na výkon.

5.2 Bezpečnostní požadavky

Obsahuje popis požadavků, zabývajících se možnými zraněními, ztrátami nebo škodami, které by mohly používáním navrhovaného systému nastat. Uvádí se zde všechna bezpečnostní opatření, které je třeba provést a všechny akce které by mohly být nebezpečné a je jim tudíž

třeba zabránit. Jsou zde zapsány certifikace, předpisy a nařízení, kterým systém musí odpovídat. Příkladem takových bezpečnostních požadavků jsou:

BEZ – 1 Pokud tlak v nádrži přesáhne 95% maximálního předepsaného tlaku, systém do jedné sekundy ukončí jakoukoliv rozpracovanou operaci.

BEZ – 2 Radiační štít smí být otevřen jen pod dohledem počítače. Pokud se spojení s počítačem z jakýchkoliv důvodů přeruší, štít se musí automaticky uzavřít.

Dále je nutné popsat veškeré požadavky na bezpečnost, integritu a důvěrnost přístupu k systému a datům, která systém zpracovává nebo vytváří, jako například:

BEZ – 3 Každý uživatel si okamžitě po prvním úspěšném přihlášení musí změnit heslo. Opětovné použití prvního hesla je zakázáno.

BEZ – 4 Po úspěšném přečtení bezpečnostního tokenu budou dveře otevřeny na 8 sekund.

Ukázka ze specifikace vyvíjeného systému:

- **BEZ-1** pro přístup do systému se musí každý uživatel přihlásit pod svým uživatelským jménem.
- **BEZ-2** každý uživatel má přidělenu roli podlé které smí přistupovat jen k funkcím, které má povoleny.

5.3 Kvalitativní parametry

Popisuje ostatní důležité kvalitativní požadavky. Měly by být konkrétní, měřitelné a ověřitelné, navíc by měly být rozděleny dle priorit (vysoká použitelnost může být důležitější než snadné učení). Dobře požadovanou úroveň kvality zachytí specializovaný zápis, jako je například jazyk Planguage.

Ukázka ze specifikace vyvíjeného systému:

Z pohledu vývojáře:

- Udržovatelnost
    - Systém musí být vhodně popsán komentáři, aby v něm byla jednodušší orientace.
    - K systému musí být sepsána podrobná dokumentace implementace.

6 Ostatní požadavky

Popisuje libovolné další požadavky, které nebyly popsány v předešlých částech specifikace, může se jednat například o právní požadavky, požadavky na lokalizaci atd. Dále je možno přidat část, která se věnuje provozu, správě a údržbě systému. Ta může popisovat instalaci systému, nastavení, spouštění a ukončení, dále opravu chyb a odolnost proti výpadkům.

Je – li ve specifikaci potřeba nějakých částí navíc, může si návrhář šablonu specifikace podle potřeby upravit.

Dodatek A: Slovníček

Ve slovníčku pojmů jsou definovány všechny výrazy, které čtenář při čtení specifikace bude potřebovat. Patří sem také zkratky, jejich definice a vysvětlení.

Dodatek B: Analytické modely

Tato část specifikace je nepovinná. Obsahuje nebo se odkazuje na analytické modely související s popisovaným systémem. Jedná se o diagramy datových toků, diagramy tříd, přechodové diagramy či ER diagramy.

Dodatek C: Seznam úkolů

Tento dodatek je také nepovinnou částí specifikace. Slouží jako průběžně aktualizovaný seznam otevřených požadavků, které je třeba dokončit (např. chybějící informace v požadavcích, čekání na něčí rozhodnutí, řešení konfliktu). [4]

5 Analýza systému

Analýza systému je jednou z fází životního cyklu softwaru. Jedná se o část postupu vývoje informačního systému. V analýze je systém popsán z různých hledisek, dekomponován na menší části, například na jednotlivé bloky plnící určité funkce. Jejím úkolem je formulovat požadované funkce systému. Toto slouží pro jednodušší pochopení vyvíjeného systému předtím, než je systém implementován. Jednotlivé diagramy mohou popisovat systém například podle toho, jak spolu mezi sebou jednotlivé komponenty komunikují nebo jen graficky znázorňují určitou funkci systému. Existují 2 různé způsoby analýzy systému a to strukturovaná analýza a objektově orientovaná analýza. Tyto části budou přiblíženy v následujících podkapitolách.

5.1 Strukturovaná analýza systému

Strukturovaná analýza obvykle vytváří hierarchické uspořádání jednotného abstraktního mechanismu. Může využívat například rodinu modelovacích jazyků IDEF (Integration DEFinition).

5.1.1 Popis strukturované analýzy

Jedná se o řízený proces začínající požadavkem a konkrétní pohled na něj. Tato metoda identifikuje celkovou činnost a tu opakovaně dělí na menší bloky činností až na úroveň jednotlivých funkcí. Při tomto postupu musí být zachovány vstupy, výstupy, ovládací prvky a mechanismy, potřebné pro optimalizaci procesů. Tento proces je také znám jako *funkční dekompozice,* zaměřena na soudržnost funkcí a vazbu mezi funkcemi vedoucími k strukturovaným datům.
Funkční dekompozice strukturované metody popisuje proces bez vymezení chování systému a udává strukturu systému ve tvaru požadované funkce. Metoda označuje vstupy a výstupy jako vztahy k aktivitě. Jeden z důvodů popularity strukturované analýzy je její intuitivní schopnost ke komunikaci procesů vysoké úrovně a systémů.

Strukturovaná analýza nahlíží na systém z perspektivy toku dat systémem. Činnost systému je popsána pomocí procesů transformujících toky dat. Strukturovaná analýza využívá výhodu skrývání informace skrz analýzu postupnou dekompozicí. To umožňuje soustředění se na detaily v jednotlivých úrovních dekompozice a vyvarovat se soustředění pozornosti k nepodstatným detailům ve vyšších úrovních. Se snižující se úrovní dekompozice se snižuje tolerance kladená na detail. Výsledkem strukturované analýzy je sada souvisejících grafických diagramů, popisů procesů a definicí dat. Popisují transformace, které potřebují k uskutečnění a požadovaná data ke shromáždění systémových funkčních požadavků. [12]

5.1.2 Prvky strukturované analýzy

Součástí strukturované analýzy jsou (dle De Marcova přístupu):

- kontextový diagram,
- dataflow diagram,
- specifikace procedu,
- Datový slovník. [12]

Přičemž kontextový diagram je diagramem na nejvyšší úrovni analýzy (nultá úroveň). Nižší úrovně jsou dále realizovány dataflow diagramy, které jsou příznačně číslovány podle nadřazené funkce (1, 1.1, 1.1.1, atd.), ke které daná nižší funkce náleží.

Kontextový diagram

Kontextové diagramy jsou diagramy reprezentující veškeré externí entity, které mohou se systémem komunikovat či jej nějakým způsobem ovlivňovat. Je to nejvyšší úroveň pohledu na systém (zvláštní typ dataflow diagramu), podobný blokovému diagramu, ukazuje základ systému znázorněný jako jeden proces (celek) a jeho vstupy a výstupy od nebo k vnějším terminátorům.
Proces ve středu diagramu zobrazuje popisovaný systém bez detailů vnitřní struktury, okolo něj obklopují všechny systémy, se kterými systém komunikuje, zařízení a činnosti. Cílem kontextového diagramu je zaměřit pozornost na externí faktory a události, které by měly být zvažovány ve vývoji kompletní sady systémových požadavků a omezení. [12]

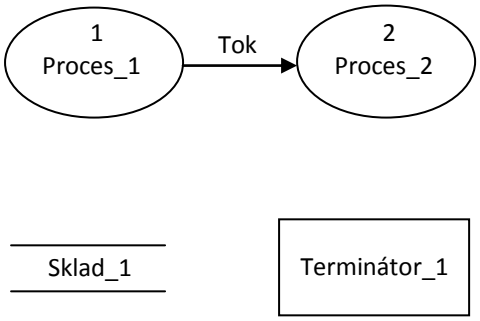
Data Flow diagram

Patří mezi nástroje sloužící k modelování funkcí systémů. Data flow diagram se skládá z procesů, toků, skladů a terminátorů.
Proces – představuje část systému, která mění vstupy na výstupy. Znázorňuje se jako kruh, ovál, obdélník nebo obdélník se zaoblenými rohy. Způsob zápisu závisí od typu zvolené notace. Název procesu je krátký a výstižný. Musí vyjadřovat podstatu procesu.
Tok – znázorňuje přesun informací (dat) z jedné části systému do druhé. Znázorňuje se šipkou (ta udává směr toku). Není – li díky propojeným entitám jasné, jaké informace jsou přenášeny, je třeba toky pojmenovat. Z jejich názvu musí být druh informací zřetelný. Tok má přenášet jen jeden druh informací. Propojuje procesy, skladiště a terminátory.
Sklad – slouží k uložení dat k jejich pozdějšímu použití. Znázorňují se dvěma vodorovnými čarami. Název je odvozen od vstupních a výstupních toků ze skladu, bývá to podstatné jméno v množném čísle.
Terminátor – představuje externí entity, komunikující se systémem. Tyto entity jsou vně systému (organizace, skupiny lidí atd.). [12]

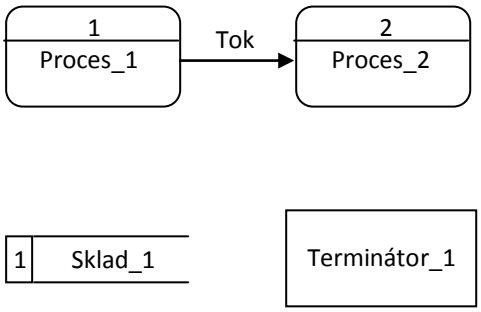
Notace DFD:

Notace udává způsob zápisu prvků diagramů. Patří mezi ně Yourdonova notace a notace Gane & Sarson.

Yourdonova notace:



Notace Gane & Sarson:



Základní pravidla pro tvoření DF diagramů jsou:

- Názvy entit by měly být srozumitelné i bez dalších poznámek či vysvětlivek (pro odborníky i pro laiky), měly by být obecné a i tak přesně specifikovat danou entitu.
- Pro jednodušší orientaci v diagramu a odkazování na konkrétní procesy by měly být procesy očíslované. Číslování je náhodné, napříč úrovněmi DFD je však nutné konzistenci (posloupnost) číslování zachovat.
- Maximální počet procesů v jednom DFD by měl být 6 až 9.
- Minimální počet procesů v jednom DFD je 3. [13]

Specifikace procesu

Specifikace procesu se může skládat z pseudokódu, vývojového diagramu nebo strukturované angličtiny. [12]

Datový slovník

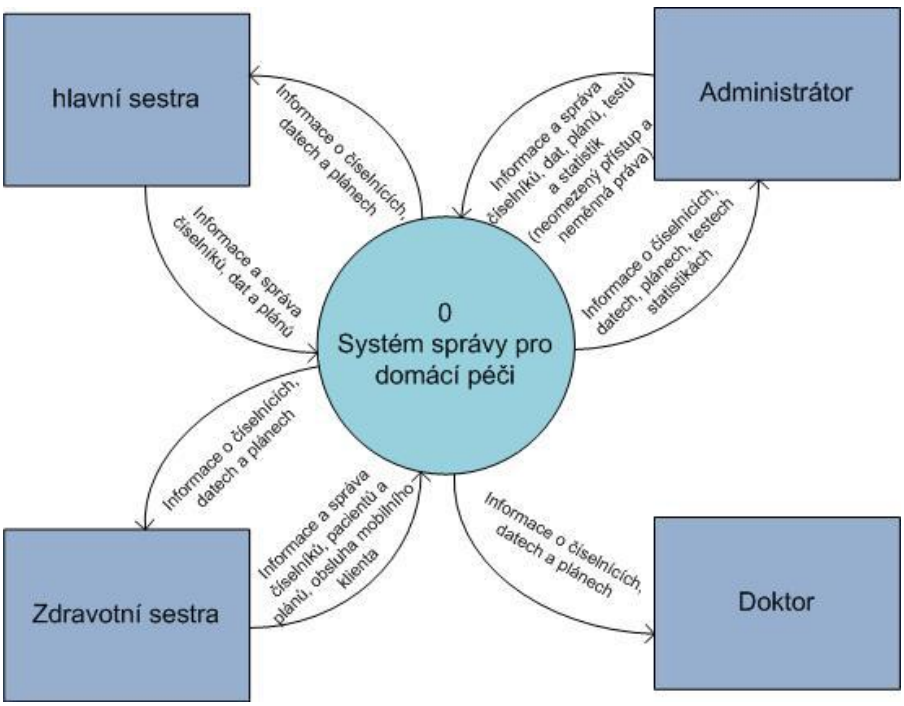
Datový nebo databázový slovník je soubor definující základní organizaci databáze. Databázový slovník obsahuje seznam všech souborů v databázi, počet záznamů v každém souboru a název a typ každého datového pole. Většina databázových systémů řízení udržuje datový slovník před uživateli schovaný, aby ho uchránili před náhodným zničením jeho obsahu. Datové slovníky neobsahují aktuální data z databáze, pouze účetní informace pro jejich správu. Bez datového slovníku však databázový systém řízení nemůže získávat data z databáze. [12]

5.2 Strukturovaná analýza vyvíjeného systému

Tato diplomová práce se zabývá převážně objektově orientovanou analýzou. Pro ukázku a srovnání, však uvedu analýzu vyvíjeného systému i z tohoto hlediska, ne však její kompletní podobu. Nejprve je proveden kontextový diagram celého biomedicínského systému (viz. Obr. 5.1), ten je pak dekomponován do data flow diagramů nulté až páté úrovně. Počet úrovní DFD je systém od systému odlišný, závisí na obsáhlosti a složitosti vyvíjeného systému.

5.2.1 Kontextový diagram systému

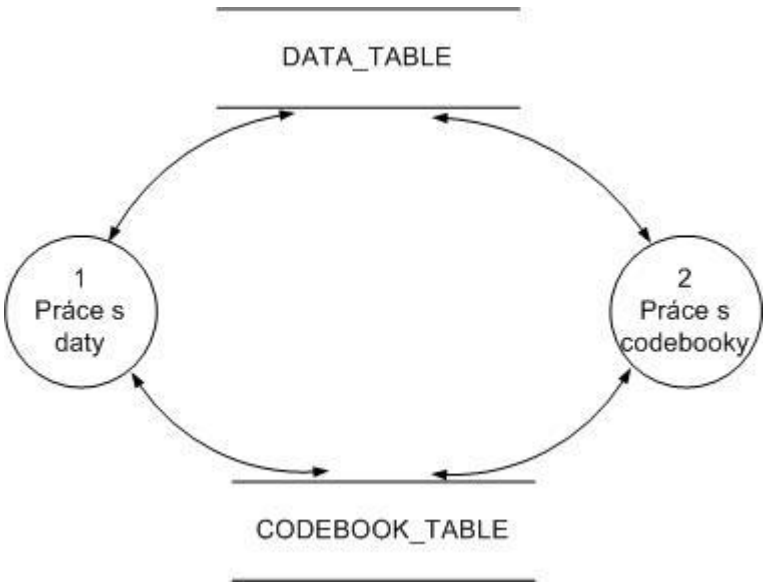
Jak již bylo uvedeno výše, kontextový diagram je nejvyšší úrovní ve strukturované analýze. Jediný proces, který diagram obsahuje, znázorňuje celý vyvíjený biomedicínský systém. Terminátory, které jej obklopují, znázorňují vnější uživatele, kteří systém nějakým způsobem ovlivňují nebo s ním komunikují. Vyvíjený systém obsahuje 4 terminátory (v případě tohoto systému zobrazuje vnější uživatele), všechny plní podobné funkce, nemají však stejná práva přístupu ke všem funkcím, které systém umožňuje. Nejnadřazenějším je *Administrátor*, ten má přístup ke všem funkcím systému, navíc je to jediný uživatel, jehož práva vůči systému jsou neměnná. Práva ostatních uživatelů mohou být měněna, tak jak jejich práva uvádí diagram (viz. Obr. 5.1), budou nastavena při dodání systému zadavateli. Data, která si systém vyměňuje s vnějšími systémy, jsou zobrazena šipkou, směr šipky udává směr toku dat (šipka směřuje k terminátoru – čtení dat systémem, šipka směřuje od terminátoru – zápis dat do systému).



Obrázek 5.1 Kontextový diagram

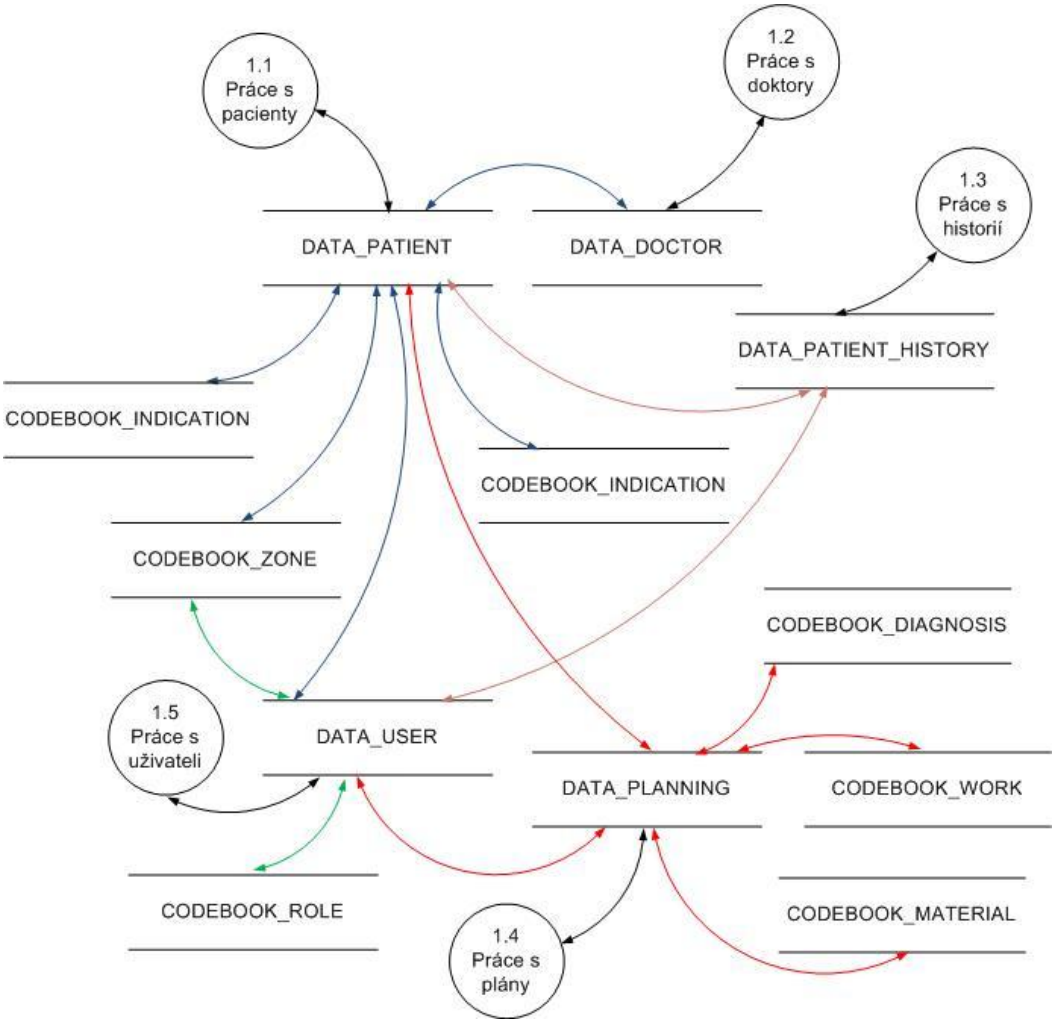
5.2.2 Data flow diagramy vyvíjeného systému

DFD nulté úrovně zobrazuje 2 základní procesy, ze kterých se celkový systém skládá. Systém se tedy skládá z procesu s názvem „Práce s daty“ a z procesu „Práce s codebooky“. První proces zpracovává data pacientů, uživatelů, pracuje s daty plánů atd. Druhý proces se zabývá zprávou veškerých číselníků, potřebných pro správnou funkci aplikace (zóny, materiály, role atd.). Tyto procesy komunikují s databázovým serverem, který uchovává veškerá data potřebná pro správu agentury.



Obrázek 5.2 DFD nulté úrovně

DFD první úrovně detailněji popisuje proces „Práce s daty“, zmiňovaný a zobrazený v DFD nulté úrovně. Tento proces se tedy dělí na procesy „Práce s pacienty“, „Práce s doktory“, „Práce s historií“, „Práce s plány“ a „Práce s uživateli“. Dále diagram zobrazuje databázové tabulky, se kterými procesy komunikují (čtou nebo zapisují informace).



Obrázek 5.3 DFD první úrovně

5.3 Objektově orientovaná analýza - UML

UML je zkratka anglického spojení Unified Modelling Language, což v českém překladu znamená Unifikovaný Modelovací Jazyk. Je to grafický jazyk, sloužící k analýze, návrhu, specifikaci či dokumentaci softwarových systémů. Grafický rozbor vyvíjených systémů je snáze pochopitelný (názornější) než textový popis, proto by alespoň základní UML analýza, jako například Diagramy datových toků, měla doprovázet každou systémovou specifikaci systému (hlavně pro lepší orientaci analytiků a vývojářů). K realizaci UML diagramů se používají takzvané Case nástroje, například Enterprise Architekt, Star UML atd.

5.3.1 Struktura jazyka UML

Struktura jazyka UML je sestavena ze stavebních bloků, společných mechanismů a architektury.

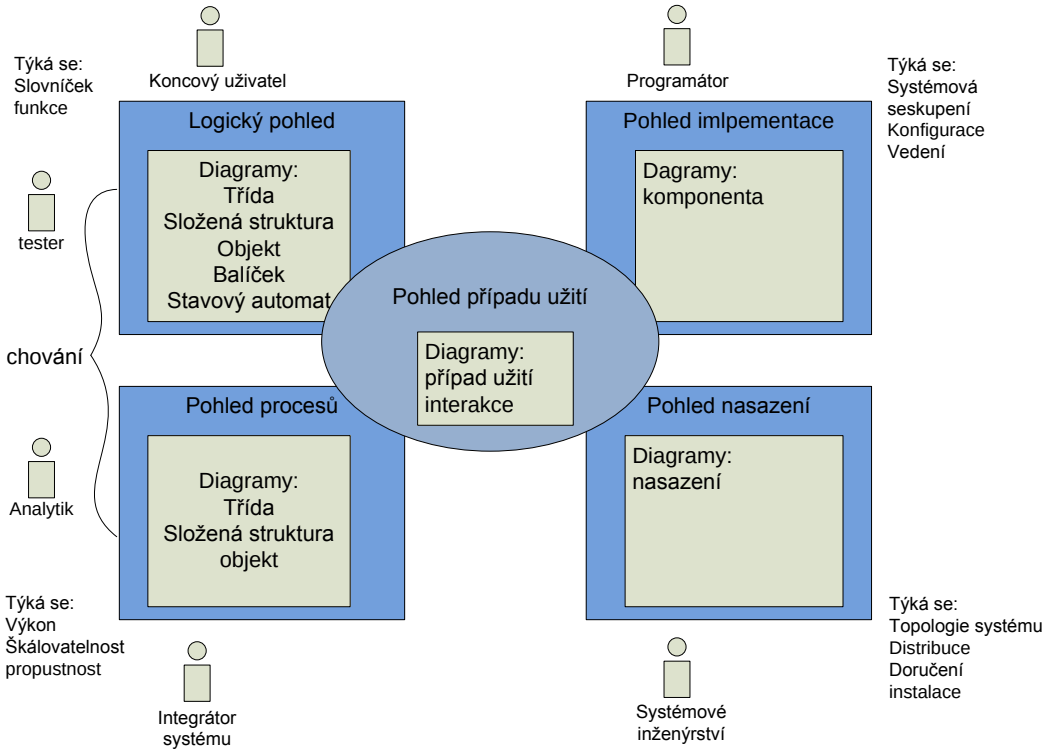
Stavební bloky - jsou základní prvky modelu, relace a diagramy.

Společné mechanismy – obecné způsoby, jimiž je v jazyce UML dosaženo specifických cílů.

Architektura

Architektura systému je definována jako „Organizační struktura systému, včetně jeho rozkladu na součásti, jeho propojitelnosti, interakce, mechanismů a směrných zásad, která proniká do návrhu systému“. Standard IEEE definuje architekturu systému jako „nejvyšší úroveň koncepce systému v jeho vlastním prostředí.“

Nejčastější pohled na architekturu je ve formě „4+1". Pro zachycení všech důležitých aspektů systému se na architekturu v jazyce UML pohlíží čtyřmi různými pohledy a to logický pohled, pohled procesů, pohled implementace a pohled nasazení. Všechny tyto pohledy jsou integrovány do jednoho nazývaného „pohled případů užití“. Znázornění pohledu na architekturu ve formě „4+1“ ukazuje obrázek 5.4. [7]



Obrázek 5.4 Architektura UML z pohledu „4+1“

5.3.2 UML diagramy

Typů UML diagramů je mnoho a každý z nich popisuje chování systému z jiného hlediska. Některé diagramy popisují funkce, které má systém mít, jiné naopak podrobně popisují chování určité funkce. Dělíme je do dvou skupin a to na diagramy chování a diagramy struktur.

Diagramy chování jsou „use case diagram“, „stavový diagram“, „diagram aktivit“, „sekvenční diagram“, „diagram komunikací“, „diagram časování“ a „diagram přehledu interakcí“.

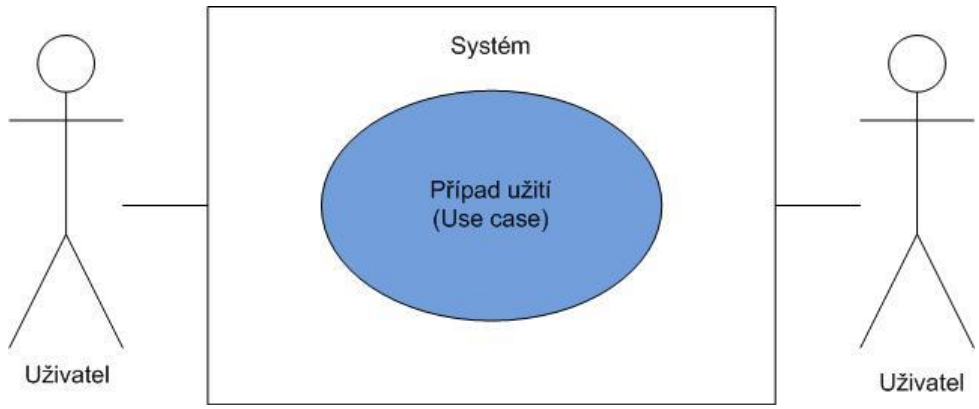
Mezi diagramy struktur patří „diagram tříd“, „diagram komponent“, „diagram nasazení“, „objektový diagram“ a „diagram balíčků“.

Následující podkapitoly popisují základní typy UML diagramů.

5.3.2.1 Use Case diagram (Diagram případů užití)

Diagram případu užití je základním modelem systému v průběhu jeho projektování a návrhu, který popisuje chování systému nebo některé jeho části z hlediska uživatele.

Díky případu užití může analytik snadno pochopit, jak má systém fungovat, navíc je z něj možné určit požadavky, které mají být na systém kladeny z hlediska uživatele. Základní strukturu Use case diagramu, vidíte na obrázku 5.5.



Obrázek 5.5 Základní struktura use case diagramu

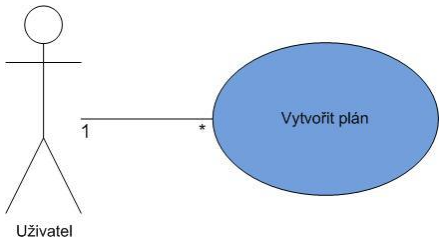
Jak je vidět na obrázku, každý use case diagram obsahuje účastník, případ(y) užití, ohraničení a relace.
Kde každý prvek plní následující funkci:

***Účastník (actor)* –** může reprezentovat osobu, HW, čidlo nebo nějaký jiný systém. Představuje prvek, který se systémem komunikuje. Účastník je oprávněn pouze k příjmu nebo odesílání informací do systému.
***Ohraničení (boundary)* –** ohraničuje systém nebo subsystém. Reprezentuje systém, jehož funkce má diagram popisovat.
***Případ užití (use case)* -** představuje službu nebo – li funkci, kterou může systém poskytovat, jinak řečeno ukazuje způsob použití systému uživatelem. Případ užití může mít svůj název a textovou specifikaci.
***Relace (relationships)* –** zobrazuje návaznosti mezi jednotlivými prvky diagramu.

Typy relací Use case diagramu:

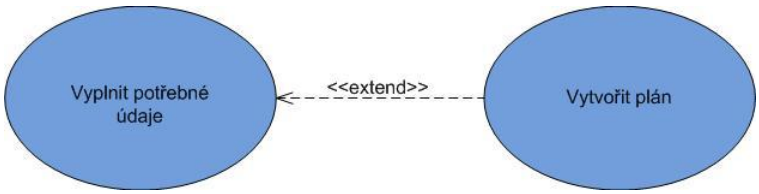
- **Asociace –** je to komunikační relace vyjadřující tok informace mezi prvkem a případem užití. Vyjadřuje se souvislou čarou mezi aktérem a případem užití, dále může tato asociace obsahovat jméno a násobnost. Násobnost je vyznačena číslem na kraji relace.

Ukázka použití asociace i s násobností je zobrazena na obrázku 5.6.



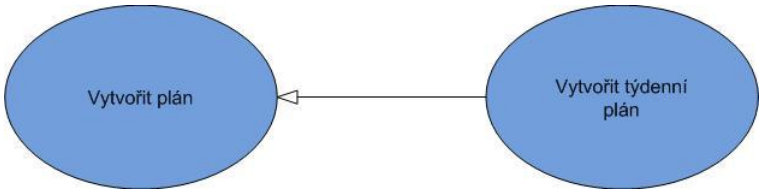
Obrázek 5.6 Ukázka použití asociace v Use case diagramu

- **Extend (rozšíření)** – označuje návaznost konkrétního případu užití s obecnějším případem užití. Vlastnosti konkrétnějšího případu užití jsou sjednoceny s vlastnostmi obecnějšího případu užití. Označuje se čárkovanou čarou zakončenou šipkou. Šipka směřuje od rozšířeného případu užití k výchozímu. Relace je popsána popisem <<extend>>. Ukázka použití relace extend je zobrazena na obrázku 5.7.



Obrázek 5.7 Ukázka použití relace extend v Use case diagramu

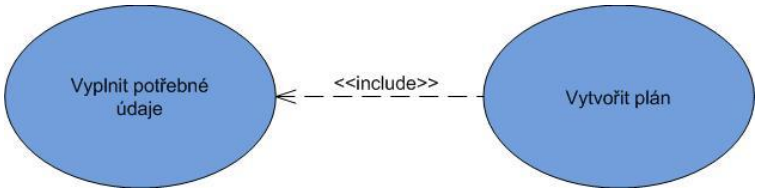
- **Generalizace (zobecnění)** – je znázorněním toho, že některý případ užití může být zobecněním jiného případu užití. Zobecněný případ užití pak nazýváme „rodič“ a specializovaný případ užití nazýváme „potomek“. Potomci dědí veškeré vlastnosti a chování svých rodičů, navíc k těmto zděděným vlastnostem mohou mít i své vlastnosti a chování. Potomek bývá upřesněním rodiče. Zobecnění používáme, je-li třeba znázornit, že některé případy užití dědí všechny vlastnosti jiného případu užití. Zobecnění je možné použít i u actorů. [5], [6], [9]



Obrázek 5.8 Ukázka použití relace generalizace v Use case diagramu

- **Include (zahrnutí) –** označuje, že některé kroky obsažené jedním případem užití mohou být použity i pro jiný případ užití. Funguje tím způsobem, že případ užití

provádí své kroky až do místa, kdy je zapotřebí vykonat kroky jiného use case. V tomto místě vykoná kroky zahrnutého use case a poté pokračuje ve vykonávání svých kroků. Kreslí se jako čárkovaná čára zakončena šipkou, ovšem na rozdíl od relace extend je popsána slovem <<include>>. Jeden případ užití může být zahrnut ve více případech užití a naopak může zahrnovat více případů užití. [5], [6], [7]



Obrázek 5.9 Ukázka použití relace include v Use case diagramu

5.3.2.2 Class diagram (Diagram tříd)

Diagram tříd zobrazuje typy objektů v podobě tříd, jejich obsah a statické vztahy mezi nimi. Z toho vyplývá, že je to takzvaný statický pohled na systém.

V diagramu tříd se mohou objevit tyto prvky:

Class (třída) **–** je popis typu objektu, může obsahovat atributy a operace (chování), které bude mít každý objekt patřící do této třídy. Třída se zobrazuje jako obdélník, jméno třídy se zapisuje velkým prvním písmenem, je – li jméno tvořeno dvěma slovy, pak se zapisuje bez mezery mezi slovy a každé další slovo názvu začíná velkým písmenem.
Atribut může obsahovat:
<<stereotyp>>viditelnost/jméno:typ násobnost = implicitní – hodnota {property string}
kde:

<<stereotyp>>	….. nepovinná část
Viditelnost	….. nepovinná část, která určuje, odkud bude prvek této třídy viditelný (public, private, protected, package).
/	….. označuje, že je atribut odvozený.
Jméno	….. povinná část, označuje jméno třídy.
Typ	….. určuje jaký typ objektu, může být v atributu pamatován.
Násobnost	….. nepovinná část, slouží k určení polí (např. [1..3] určuje, že bude složen z 1 až 3 řetězců)
Implicitní hodnota	….. nepovinná část, tato hodnota je atributu vytvořena po vzniku instance, není – li specifikována během vytváření instance
{property string}	….. nepovinná část, umožňující specifiovat další vlastnosti atributu (např. {readOnly}).

Package (balíček) **–** balíček umožňuje seskupení tříd do skupin, například v případě, že chceme upozornit na to že některé třídy náleží do určitého podsystému.

Rozhraní (interface) – rozhraní je klasifikátor, charakterizující jen ohraničenou část chování modelované entity. Slouží ke specifikaci parametrů modelu viditelných zevně bez toho, aby byla zobrazena jejich struktura. Je ekvivalencí k abstraktní třídě obsahující abstraktní operace. Rozhraní neobsahuje atributy.
Relace – popisuje návaznost mezi třídami.

Typy relací Diagramu tříd:

- Asociace

Určuje spojení mezi dvěma třídami, nebo – li vztah jedné třídy k třídě druhé (např. chceme – li zobrazit, že zdravotní sestra pracuje v agentuře spojíme asociací třídy „Zdravotní sestra“ a „Agentura“ viz. obrázek 5.10).

Obrázek 5.10 Ukázka použití asociace v diagramu tříd

Obrázek 5.10 ukazuje kromě popisu vztahu mezi dvěma třídami také roli každé z tříd, ta je napsána pod asociací u třídy, jejíž roli popisuje. Role zaměstnanec popisuje, roli zdravotní sestry vůči třídě „Agentura“ a naopak role „zaměstnavatel“ popisuje roli třídy „Agentura“ vůči třídě „Zdravotní sestra“. Čísla zobrazena nad asociací označují možnou násobnost třídy, vedle které leží. To znamená, že číslo jedna označuje, že „zdravotní sestra“ může pracovat pro jednu agenturu, číslo 1..* u třídy „zdravotní sestra“ zobrazuje, že pro agenturu může pracovat minimálně jedna a maximálně libovolné množství sester.

- Generalizace (zobecnění)

Zobecnění znázorňuje dědičnost vlastností třídy potomka od třídy rodiče. Znázornění je stejně jako u use case diagramu čarou s uzavřenou šipkou na straně rodiče.

- Agregace

Znázorňuje vztah mezi komponentou a třídou, kterou komponenta vytváří, tedy například monitor je komponentou, ze které se skládá PC klient vyvíjené aplikace. Zobrazuje se plnou čarou zakončenou nevyplněným kosočtvercem na straně třídy, vytvářené komponentou. Agregace může obsahovat násobnost, ta může znázornit, že se PC klient bude skládat např. ze 2 monitorů.

- Kompozice

Jedná se o silnější typ agregace. Každá komponenta v kompozici může náležet pouze jednomu celku, navíc dojde – li ke zničení celku, bude zničena i komponenta vytvářející celek. Označuje se v diagramu plnou čarou zakončenou kosočtvercem vyplněným černou barvou.

- Realizace

Je relace mezi rozhraním a třídou, kdy se třída chová z větší části stejně jako rozhraní. Jedna třída může realizovat více než jedno rozhraní a naopak jedno rozhraní může realizovat více než jednu třídu. Označuje se přerušovanou čarou zakončenou uzavřenou, nevybarvenou šipkou na konci u rozhraní. Příkladem rozhraní může být například klávesnice (znázorňuje třídu) a psací stroj (znázorňuje rozhraní). Klávesnice má s rozhraním shodnou vlastnost úhoz(), avšak klávesnice obsahuje navíc atributy jako je například alt(), ctrl(), atd. [5], [6], [7], [9]

5.3.2.3 State machine diagram (Stavový diagram)

Stavový diagram se používá k popisu reakce objektu na vnější působení. Toto vnější působení může být buď působení jiných objektů, nebo působení uživatele. Stavový diagram na rozdíl od jiných diagramů modelujících chování popisuje chování (změnu stavu) pouze jedné třídy, přesněji jedné instance třídy (objektu). Změny stavů jsou prováděny na základě přijetí konkrétních událostí.

Hlavními prvky stavových diagramů jsou:

Stav(state) – situace, kdy modelovaný objekt splňuje nějakou podmínku, provádí operaci, nebo čeká na událost. Stav je popisem abstraktní metatřídy používané pro modelování situací, v průběhu kterých má smysl vykonávat nějaké podmínky. Stav může být konkrétní soubor atributů třídy nebo objektu. Ne každý atribut třídy může modelovat její stav, většinou mají význam pouze takové elementy systému, vyjadřující nějaké chování třídy nebo její funkcionalitu. Stav je v UML vyjádřen zaobleným obdélníkem, ten může mít tři části a to název stavu, stavové proměnné a seznam činností. Nemusí být zobrazeny všechny tři části stavu, stačí pouze název.

Kompozitní stav – kompozitní stav má sestavy. Může se dělit na regiony, v nichž v každém regionu je aktivní vždy právě jeden stav.

Přechod (transition) – jde o spojení mezi dvěma stavy. Místo kdy stav přechází po splnění určitých podmínek z jednoho stavu do druhého. Přechod je označován plnou čarou zakončenou šipkou. Název události, která tento přechod vyvolala, je napsána nad (vedle) šipky přechodu. Není li u přechodu napsán název události, který spustí přechod, musí být z kontextu stavového diagramu jasné, po ukončení které akce bude přechod uskutečněn.

Rozloučení – označuje přechod do téhož stavu.

Strážní podmínky – strážní podmínka se v diagramu zapisuje v hranatých závorkách u přechodu. Strážní podmínka má buď hodnotu „pravda“ nebo „nepravda“. Má – li hodnotu „pravda“, může se přechod uskutečnit, má – li hodnotu „nepravda“ přechod uskutečněn nebude.

Stavový diagram může dále obsahovat takzvané pseudostavy. Pseudostav je prvek, který má sice formu stavu, ale vyznačuje se speciálním chováním. Je – li pseudostav aktivní, není ještě dokončena reakce na událost, která přechod vyvolala. Pseudostav je definice detailu přechodu.

Mezi pseudostavy patří:
Počáteční stav (initial state) – zvláštní pseudostav pro počátek automatu. Tento stav neobsahuje žádné činnosti. Objekt se zde nachází automaticky na v počáteční dobu a zobrazuje oblast, kde začíná proces změny stavu. Označuje se černě zabarveným kruhem. Šipka přechodu z tohoto stavu vždy vychází. Může také označovat implicitní stav regionu, ve kterém je obsažen. Každý region v kompozitním stavu může mít jeden počáteční stav.

Koncový stav – zvláštní pseudostav pro konec automatu. Je podobný počátečnímu stavu, s tím rozdílem, že se zde bude objekt vyskytovat pouze na konci popisu svého chování. Do tohoto stavu může šipka přechodu vždy pouze vcházet. V UML se označuje jako prázdný kruh s černě zabarveným kruhem uprostřed.

Volba (choice) – rozděluje přechod na dva segmenty. První segment je aktivován bez ohledu na další a před tím než budou vyhodnoceny strážní podmínky dalších segmentů, musí být provedeny všechny jeho efekty. Poté strážní podmínky provedou vyhodnocení s ohledem na výsledky efektů prvního segmentu. Přechod přejde do toho stavu, jehož strážní podmínka byla vykonáním efektů v prvním segmentu splněna. Nebyla li splněna ani jedna ze strážních podmínek, jedná se o špatně formovaný model. Tomuto jde zabránit tím, že jednu větev namodelujeme s podmínkou else.

Rozvětvení (fork) – používá se tam, kde se jeden stav rozvětvuje na více cílových stavů. Označuje se tlustou černou čarou.

Spojení (join) – spojení je přesný opak rozvětvení, používá se totiž tam, kde se více stavů spojuje do jednoho cílového stavu. Jeho označení je stejné jako u rozvětvení.

Historický pseudostav – indikuje, že si kompozitní stav pamatuje svůj substav, který byl aktivní při opuštění kompozitního stavu.

Ukončení (terminate) - jedná se o stav, ve kterém je ukončeno provádění stavového stroje. Ve většině případů to znamená, že vlastník stavového stroje zanikl. Označuje se křížkem. Do tohoto stavu přechod pouze vstupuje, nikdy z něj přechod nemůže vycházet. [5], [6], [9]

5.3.2.4 Activity diagram (Diagram činností)

Diagramy činností se v jazyce UML používají pro modelování procesu vykonání operací. Grafická notace diagramů činností je v mnoha věcech shodná s grafickou notací stavových diagramů. Diagramy činností totiž také znázorňují stavy a přechody mezi nimi. Přechod mezi stavy bude uskutečněn po vykonání operace stavu, ve kterém se proces nachází.
Diagram je jakýsi flowchart znázorňující tok řízení z aktivity do aktivity. Používá se mimo jiné i k modelování bussines procesů.

Hlavní prvky diagramu aktivit:

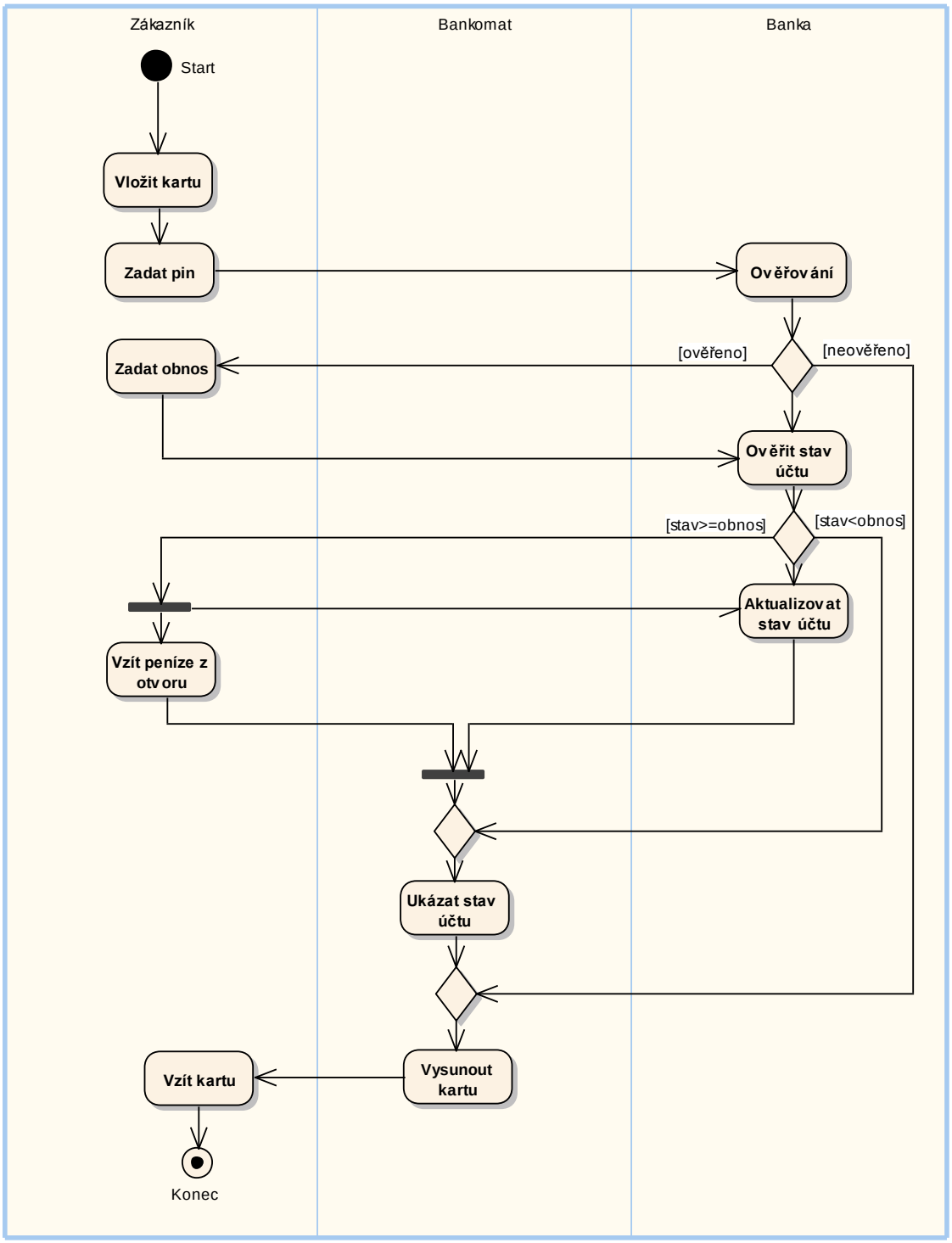
Akce (action) – nejjednodušší prvek diagramu. Nemůže již být dále dekomponován, to znamená, že znázorňuje v diagramu jednoduchou akci jako například stisk tlačítka.

Aktivita (activity) – je specifikace chování, popisuje sekvenční a souběžné kroky výpočetní procedury.

Plovoucí dráhy (swimm lines) – tento prvek je využitelný hlavně při modelování bussines procesů, kde je takto možné každou činnost přiřazovat přímo do odpovídající části podniku, ve které se činnost bude provádět. V UML jsou tyto plovoucí dráhy označovány jako svislé čáry, kdy každá dráha má svůj název (název může odpovídat například oněm jednotlivým částem podniku).

Příklad použití plovoucích drah je zobrazen na obrázku 5.11, na němž jsou zobrazeny také veškeré hlavní prvky tohoto diagramu, jako akce, rozpojení, spojení, volba (choice), sloučení (merge).

Diagram aktivit používá stejně jako stavový diagram přechody, rozhodování, strážní podmínky, rozdělení (fork) a spojení (join), proto zde již tyto prvky nebudou znovu popsány (viz. Stavový diagram výše). [5], [6]



Obrázek 5.11 Ukázka aktivity diagramu

5.3.2.5 Sequence diagram (Sekvenční diagram)

Sekvenční diagramy se používají pro znázornění vzájemného působení objektů systému nebo jeho části v závislosti na čase. V diagramu jsou zobrazeny pouze ty objekty, které spolu vzájemně komunikují a neprokazují možné statické asociace s jinými objekty.
Sekvenční diagram má 2 rozměry. První rozměr je v horizontálním směru (zleva doprava) a je znázorněn svislými přerušovanými čarami (životočarami). Druhý rozměr diagramu má svislý směr a reprezentuje časovou osu od vzniku až po zánik objektu.

Sekvenční diagram tedy obsahuje tyto prvky:

Objekt – Objekt je na životočáře zobrazen vždy na začátku (nad životočarou) a to obdélníkem obsahujícím název objektu a jméno třídy oddělené dvojtečkou. Celý název objektu je podtržen, to označuje, že jde o objekt (instanci třídy). Objekty se na diagram kreslí zleva doprava a to tak, že krajní objekt vlevo je iniciátor vzájemného působení, vpravo od něj je pak objekt, který s tímto objektem bezprostředně komunikuje. Z tohoto důvodu mají všechny objekty v diagramu uspořádání určené stupněm aktivity objektu při vzájemném působení.

Životočára – svislá přerušovaná čára znázorňující objekt od jeho vzniku až po zánik. Počáteční časový okamžik znázorňuje vrchní bod životočáry (je to místo vzniku objektu). Existuje – li objekt v systému stále, musí jeho životočára pokračovat po celé ploše diagramů odshora až dolů. Není – li již některý objekt zapotřebí a je třeba uvolnit zdroje, které využívá, například pro vznik jiného objektu, je třeba jej v časovém okamžiku, kdy jej již není potřeba, odstranit. To se provede ukončením životočáry zakončené křížkem.

Fokus řízení (focus of kontrol) – slouží k detekci aktivního stavu objektu. Objekty mohou v aktivním stavu vykonávat úlohy nebo jen pasivně očekávat zprávy od jiných objektů. V UML se označuje jako protáhlý úzký obdélník místo životočáry. Jeho vrchní strana značí začátek fokusu řízení daného objektu a spodní strana konec fokusu řízení. Objekt může mít několik fokusů řízení a to z toho důvodu, že se může měnit perioda aktivity fokusu s periodou pasivity a očekávání.

Obrázek 5.12 Ukázka objektu s životočárou a fokusem řízení

Zpráva – vede od životočáry jednoho objektu k životočáře objektu druhého. Prostřednictvím zprávy si objekty mezi sebou vyměňují informace (představuje ukončený fragment informace, který je posílán jedním objektem druhému). Zprávy nemusí jen přenášet informaci, ale také mohou požadovat po přijímajícím objektu vykonání očekávaných operací, parametry operací se předávají spolu se zprávou. V případě sekvenčního diagramu jsou všechny zprávy uspořádané v čase, tudíž zpráva vzniklá v modelovaném systému dřív, bude v diagramu zobrazena výš než zpráva vzniklá později. U sekvenčního diagramu rozeznáváme tři typy zpráv:

- jednoduchá 

– přechod řízení z jednoho objektu na druhý

- synchronní 

– objekt pošle zprávu druhému a nečinně čeká, dokud nedostane odpověď

- asynchronní 

– objekt pošle zprávu druhému objektu a pracuje dál.

Doba předání zpráv musí být dost malá ve srovnání s procesy vykonávání operací objekty. Během předání zprávy nemůže probíhat žádná jiná událost. Nemůže – li být toto dodrženo, pak se šipka nezobrazuje vodorovně, ale se sklonem (konec šipky je níž, než její začátek). V některých situacích může objet posílat zprávu i sám sobě, v tomto případě šipka nesměřuje k jinému objektu, ale vychází i směřuje sama na sebe. Tyto zprávy nazýváme reflexní.

Chceme – li zapsat složitější logiku vzájemného působení, můžeme toho docílit použitím **Rozdělení toku řízení** – znázorňuje se jako dvě nebo více šipek, vycházejících z jednoho bodu fokusu řízení. Potřebné podmínky, musí být přehledně zapsány vedle každé větve. Podmínky se vždy musí navzájem vylučovat. [6], [9]

5.3.2.6 Component diagram (Diagram komponent)

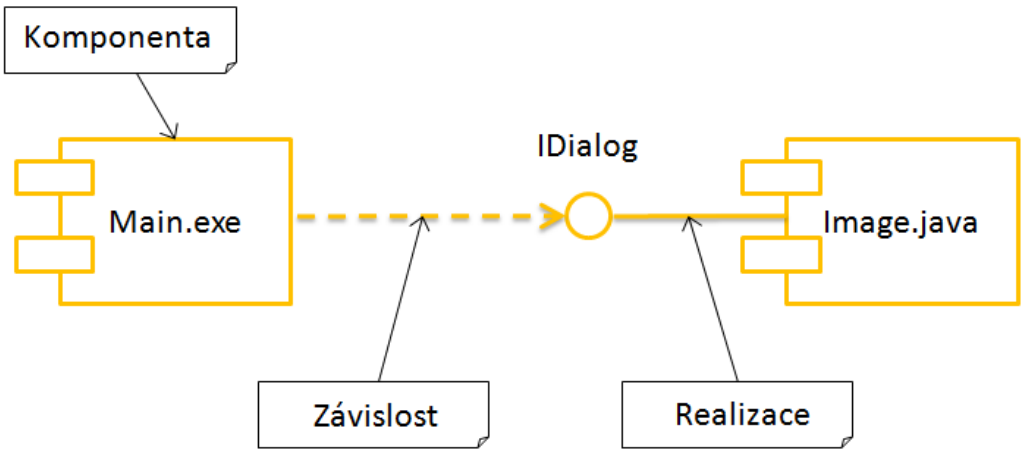
Diagram komponent popisuje vlastnosti fyzické prezentace systému. Pomáhá určit architekturu vyvíjeného systému pomocí závislosti mezi komponenty softwaru, včetně zdrojových kódů, binárních knihoven i spustitelných programů.
Hlavními prvky diagramu jsou:
Komponenty – používají se pro představení fyzických podstat v UML. Znázorňují se obdélníkem se dvěma malými obdélníky umístěnými do levé hrany hlavního obdélníku. Uvnitř hlavního obdélníku je zapsáno jméno komponenty (obvykle jména spustitelných souborů se zobrazenou příponou např. EXE), je možno připsat i nějaké doplňující informace (název balíčku, verze komponenty atd.). Při modelování diagramu komponent se můžeme setkat se třemi typy komponent:

- Rozmisťované – tvoří základ spustitelných systémů (Dll, ActiveX, Java Beans).
- Podpůrné komponenty – základ pro rozmísťované komponenty (datové soubory, soubory se zdrojovým textem).
- Prováděcí komponenty – vznikají za běhu systému

Komponenty mohou být také specifikovány podle jejich stereotypu zapsaného před jménem komponenty. V UML se pro komponenty používají následující stereotypy:

- Knihovna (library) – prezentován dynamickou či statickou knihovnou.
- Tabulka (table) – prezentován tabulkou či databází.
- Soubor (file) – prezentován soubory se zdrojovými texty programů.
- Dokument (dokument) – prezentován dokumenty.
- Spustitelný (executable) – může se vykonávat v uzlu.

Rozhraní – jeho přítomnost u komponenty znamená, že komponenta realizuje daná rozhraní. Je zakreslováno malou kružnicí spojenou plnou čarou s komponentou. Název rozhraní musí vždy začínat písmenem „I“, zapisuje se vedle kružnice znázorňující rozhraní. Rozhraní může být taky zobrazeno jako obdélník znázorňující třídu se stereotypem rozhraní, případně s oddíly pro atributy a operace. Tato varianta se používá, je – li pro realizaci potřeba znázornit vnitřní strukturu rozhraní.
Závislosti – slouží pro zobrazení vztahu, kdy změna jednoho prvku modelu způsobuje změnu druhého prvku modelu. Zobrazuje se přerušovanou čarou začínající u klienta (závislý prvek) s šipkou směřující ke zdroji (nezávislý prvek). Závislost může spojovat komponentu a rozhraní importované touto komponentou (šipka směřuje od komponenty klienta k importovanému rozhraní – v tomto případě šipka rozhraní nerealizuje, nýbrž jej používá v procesu svého vykonávání), nebo různé druhy komponent mezi sebou. Závislosti také mohou představovat závislost mezi komponentami a třídami existujícími v diagramu (z důvodu, že změny v struktuře popisu tříd mohou způsobit změnu komponenty). [6], [9]



Obrázek 5.13 Ukázka závislosti v diagramu komponent

5.3.3 UML Analýza vyvíjeného biomedicínského systému

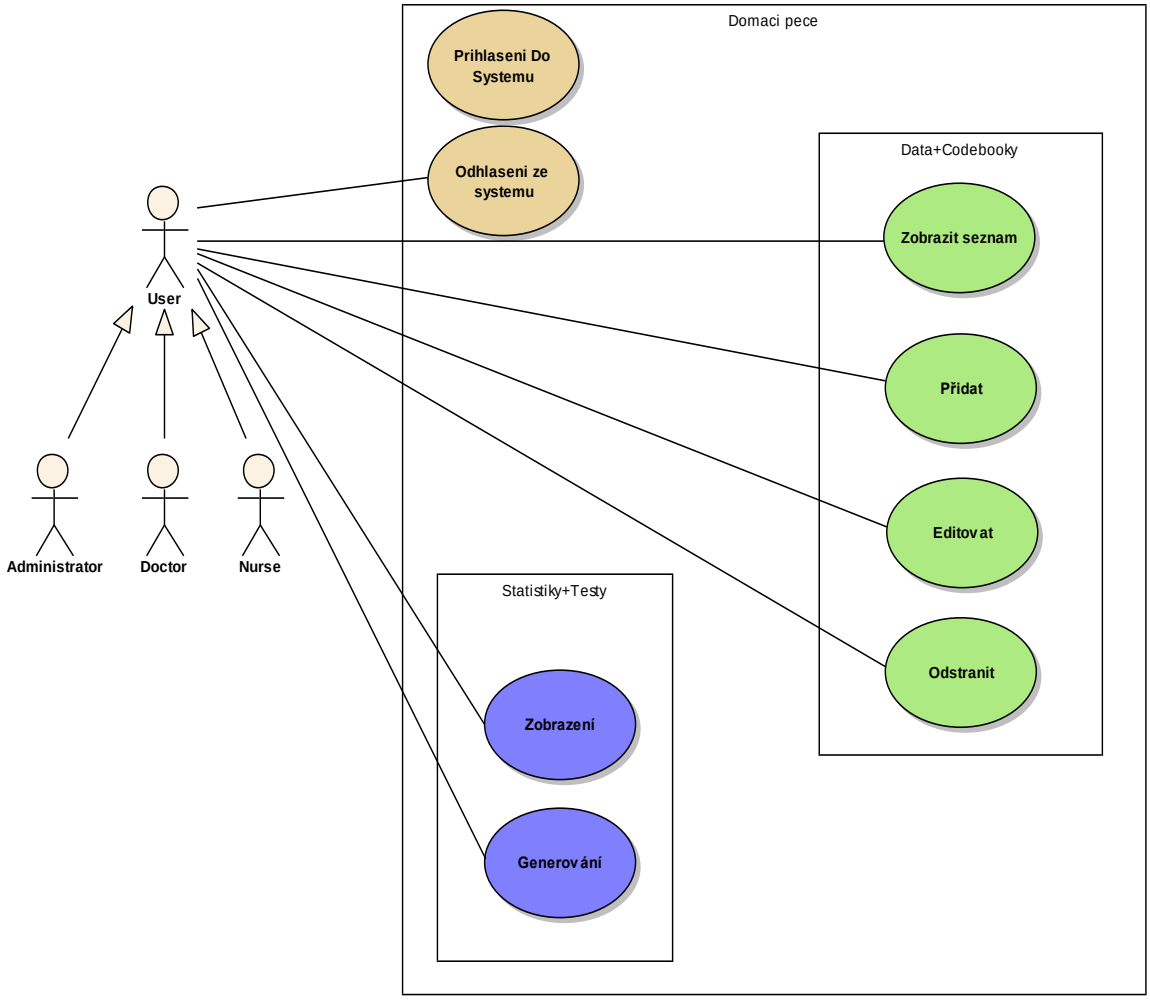
Vyvíjený systém je analyzován pomocí UML diagramů z důvodu lepší orientace a pro jednodušší pochopení požadované funkčnosti pro následnou implementaci. Struktura aplikace bude pro srovnání popsána různými typy diagramů struktur (diagram tříd, komponent, nasazení, balíčků nebo objektový diagram viz. kapitola 5.2). Systém bude také zobrazen v podobě kontextového diagramu od úrovně nula až po úroveň, která již nepůjde dále dekomponovat. Celková funkcionalita systému bude popsána use case diagramem včetně scénářů a jednotlivé use casy poté rozebrány podrobněji některým z diagramu chování. Diagramy jsou realizovány v programu Enterprise architekt 6.5 popřípadě Microsoft office Visio 2007. Z důvodu obsáhlosti kompletní analýzy budou zde zobrazeny jen některé diagramy, kompletní analýza bude dodána v přílohách diplomové práce.

5.3.3.1 Use case diagram systému

Use case diagram zobrazuje veškeré funkce, které je možno provádět v systému domácí péče. Tento systém mohou obsluhovat 2 typy aktérů a to „Administrator“ a „User“, kde druhý uvedený je zobecněním uživatelů s proměnnými právy (Nurse a Doctor). Kromě dvou hlavních funkcí v systému, což je přihlášení a odhlášení, jsou funkce prováděné nad systémem velmi podobné, proto jsou pro přehlednost diagramu tyto funkce rozděleny do skupin podle toho, nad jakými daty jsou prováděny. Tyto skupiny jsou „Data+Codebooky“ a „Statistiky+Testy“. První skupiny obsahuje případy užití zobrazit seznam, přidat, editovat a odstranit, druhá skupina obsahuje případy užití zobrazení a generování.

Při prvním pohledu na diagram, by se mohlo zdát, že mají všichni uživatelé přístup ke všem funkcím systému, rozdíl je však v přístupu uživatelů k funkcím. Administrátor má totiž přístup ke všem funkcím vždy a toto se nikdy nezmění, Po přihlášení aktéra „User“ je danému uživateli podle jeho přihlašovacích údajů v databázi přiřazená role, kterou má v systému přiřazenu a

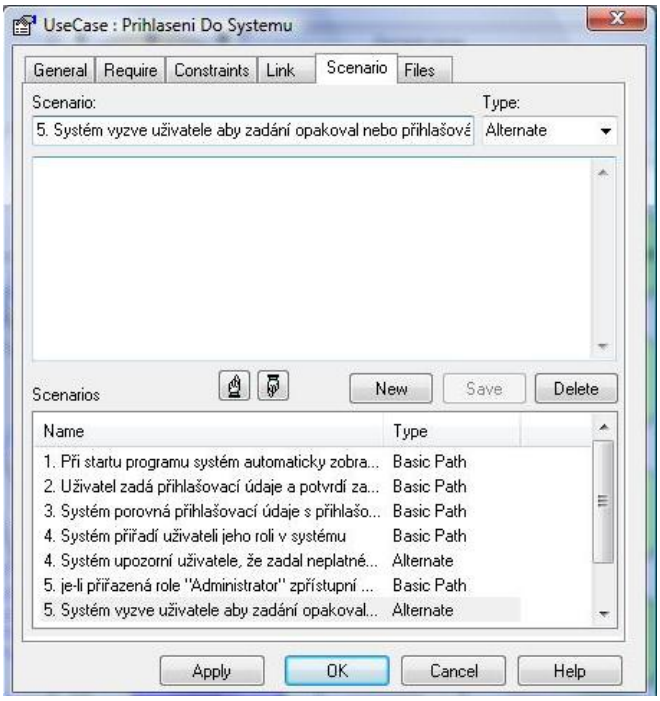
podle nastavení této role jsou přihlášenému uživateli zobrazeny pouze ty funkce, které může obsluhovat. Tento rozdíl, je popsán ve scénářích diagramu (obr. 5.14).



Obrázek 5.14 Use Case diagram vyvíjeného systému

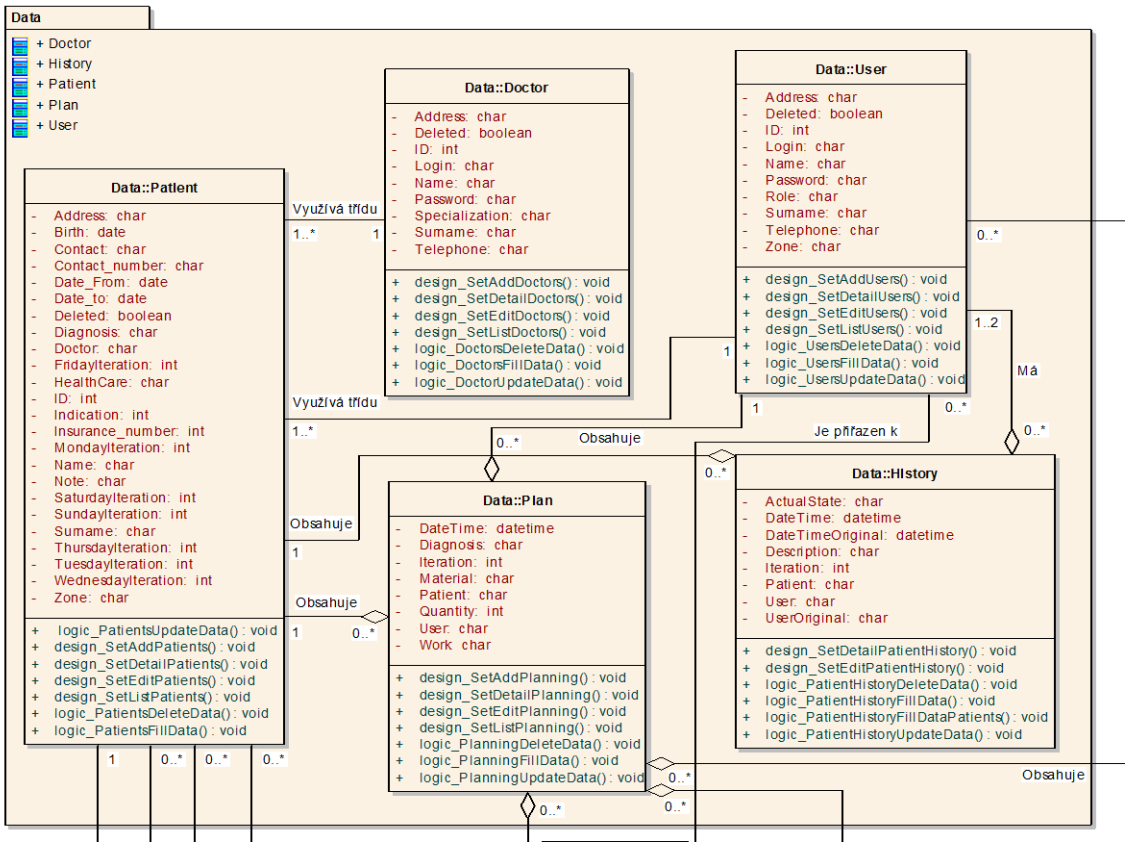
Obrázek 5.15 zobrazuje scénář případu užití „Prihlaseni do systemu“. Ten obsahuje hlavní a alternativní scénář. Ve hlavním scénáři systém porovnává údaje, které zadá uživatel do okna pro přihlášení s údaji, které jsou uloženy v databázi. Poté je uživateli přidělena buď role a podle ní jsou zpřístupněny buď všechny funkce systému (pro administrátora) nebo jen některé (podle práv přidělených dané roli). Vedlejší scénář popisuje případ, kdy údaje zadané přihlašovanou osobou nesouhlasí s údaji v databázi. V tomto případě systém vyzve uživatele, aby se pokusil přihlásit opětovně nebo aplikaci ukončil.

Ostatní scénáře pro tento diagram jsou vloženy v příloze číslo III Analytické diagramy.



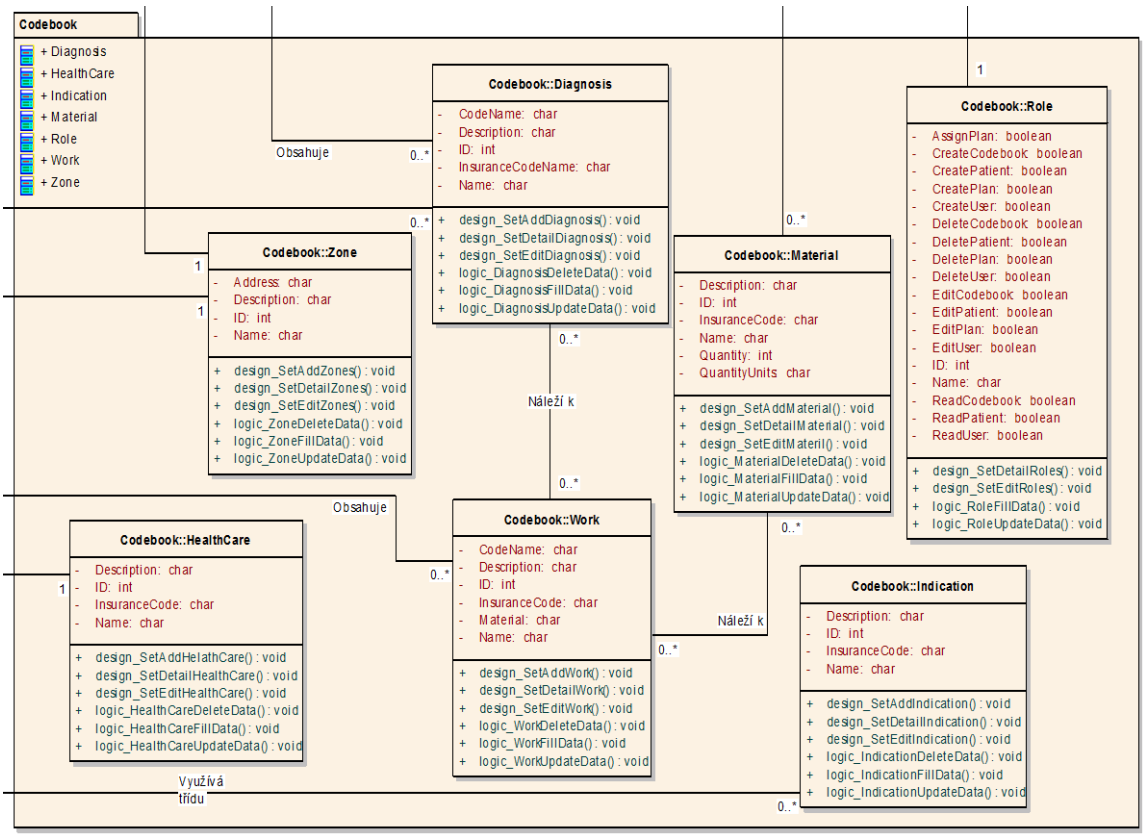
Obrázek 5.15 Scénář pro přihlášení do systému

5.3.3.1 Class diagram vyvíjeného systému



5.16 Balíček Data Class diagramu

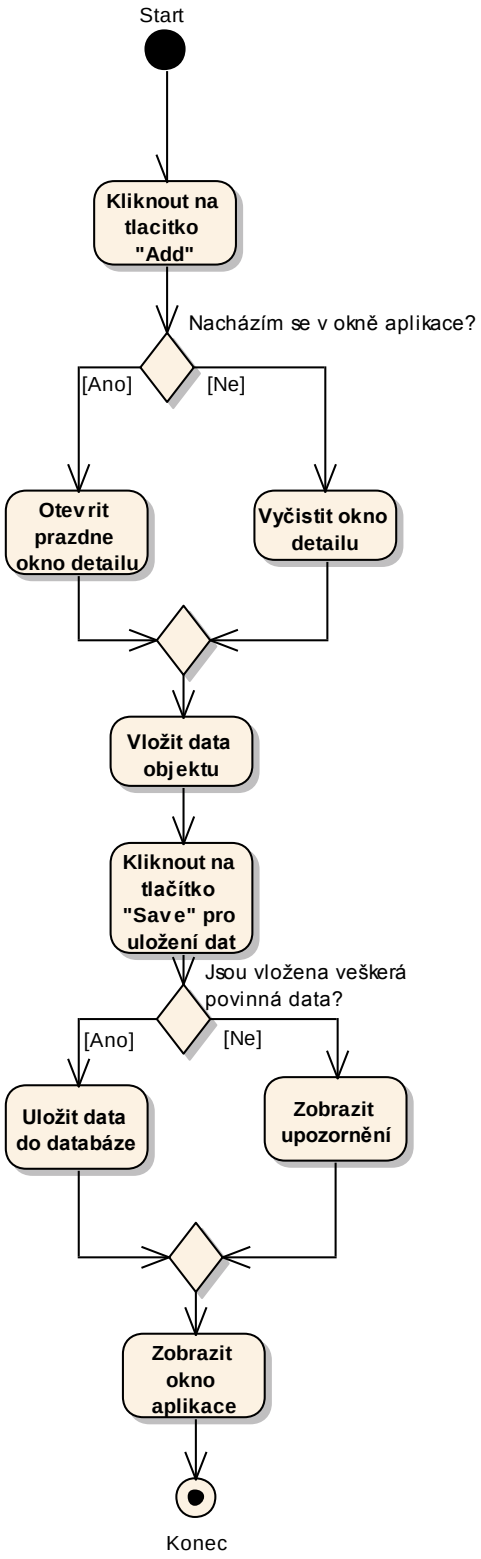
Balíček „Data“, zobrazen na obrázku 5.16, obsahuje veškeré třídy pracující s daty. Každá z těchto tříd obsahuje atributy a metody, se kterými pracuje. Třídy jsou mezi sebou relacemi propojeny, tak jak spolu komunikují nebo spolu souvisejí nějakým jiným způsobem (např. jsou svou součástí). Třídy nejsou propojeny jen v rámci jednoho balíčku, ale i mezi balíčky. Každá relace mezi třídami obsahuje kromě názvu také násobnost mezi propojenými třídami. Jako příklad uvedu propojení tříd „Patient“ a „History“. Ty jsou propojeny relací typu „Agregace“, to znamená, že třída „Patient“ je součástí třídy „History“. Násobnost a název této relace říká, že každý objekt třídy pacient může obsahovat 0 – libovolný počet objektů typu historie a v opačném směru může každý objekt typu „History“ obsahovat pouze jediný objekt typu „Patient“.



5.17 Balíček Codebook Class diagramu

Obrázek 5.17 zobrazuje druhou část class diagramu a to balíček „Codebook“. Ten obsahuje třídy pracující s daty číselníků, stejně jako v balíčku „Data“, i zde každá třída obsahuje své atributy a metody. Příkladem propojení dvou tříd, obsažených v rozdílných balíčcích, je relace mezi třídou „Zone“ z balíčku „Codebook“ a třídou „Plan“ z balíčku „User“. V tomto případě se jedná o relaci typu „Asociace“. Její násobnost a název říká, že každý objekt typu „User“ je přiřazen k právě jednomu objektu typu „Zone“ nebo – li, každý objekt typu „Zone“ má přiřazen libovolný počet objektů typu „User“. Celý diagram je zobrazen v příloze číslo III Analytické diagramy.

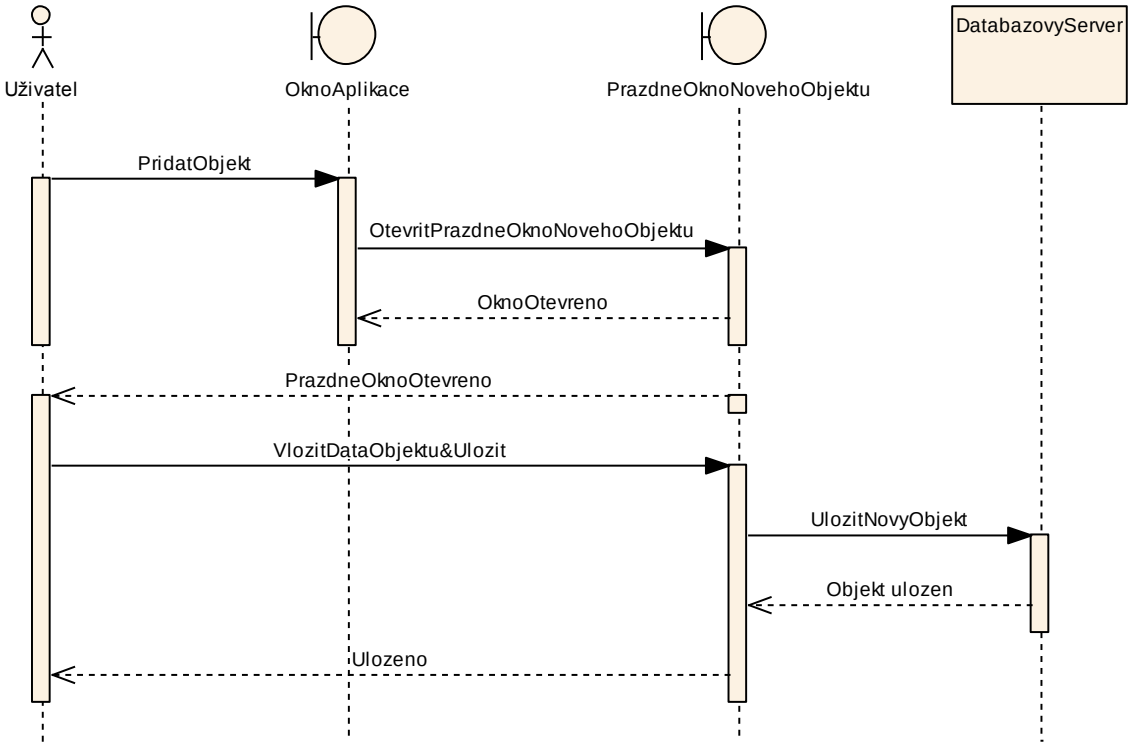
5.3.3.2 Activity diagram vyvíjeného systému



5.18 Aktivity diagram přidání objektu

Aktivity diagram na obrázku 5.18 popisuje způsob páce programu při přidání jakéhokoliv typu objektu z hlediska akcí programu. Jak je vidět na obrázku, tato část programu se spustí akcí „kliknout na tlačítko Add“ (zda je toto tlačítko v danou chvíli přístupné, je řešeno v use case diagramu a také v systémové specifikaci). V dalším kroku bude podle aktuální pozice uživatele v programu, buď otevřeno nevyplněné okno detailu objektu, nebo nachází – li se uživatel již v okně detailu nějakého objektu, bude toto okno vyprázdněno a připraveno k zadání údajů nového objektu. Poté jsou do okna vložena data nového objektu a uživatel klikne na tlačítko uložit. Pokud uživatel zadal všechny povinné položky, bude nový objekt uložen do databáze, pokud budou některá data chybět, bude uživateli zobrazeno upozornění, že nezadal veškerá potřebná data. Po úspěšném uložení nového objektu, bude zobrazeno aktualizované okno aplikace (seznam objektů) obsahující právě uložený objekt.

5.3.3.3 Sequence diagram vyvíjeného systému



5.19 Sekvenční diagram pro přidání nového objektu

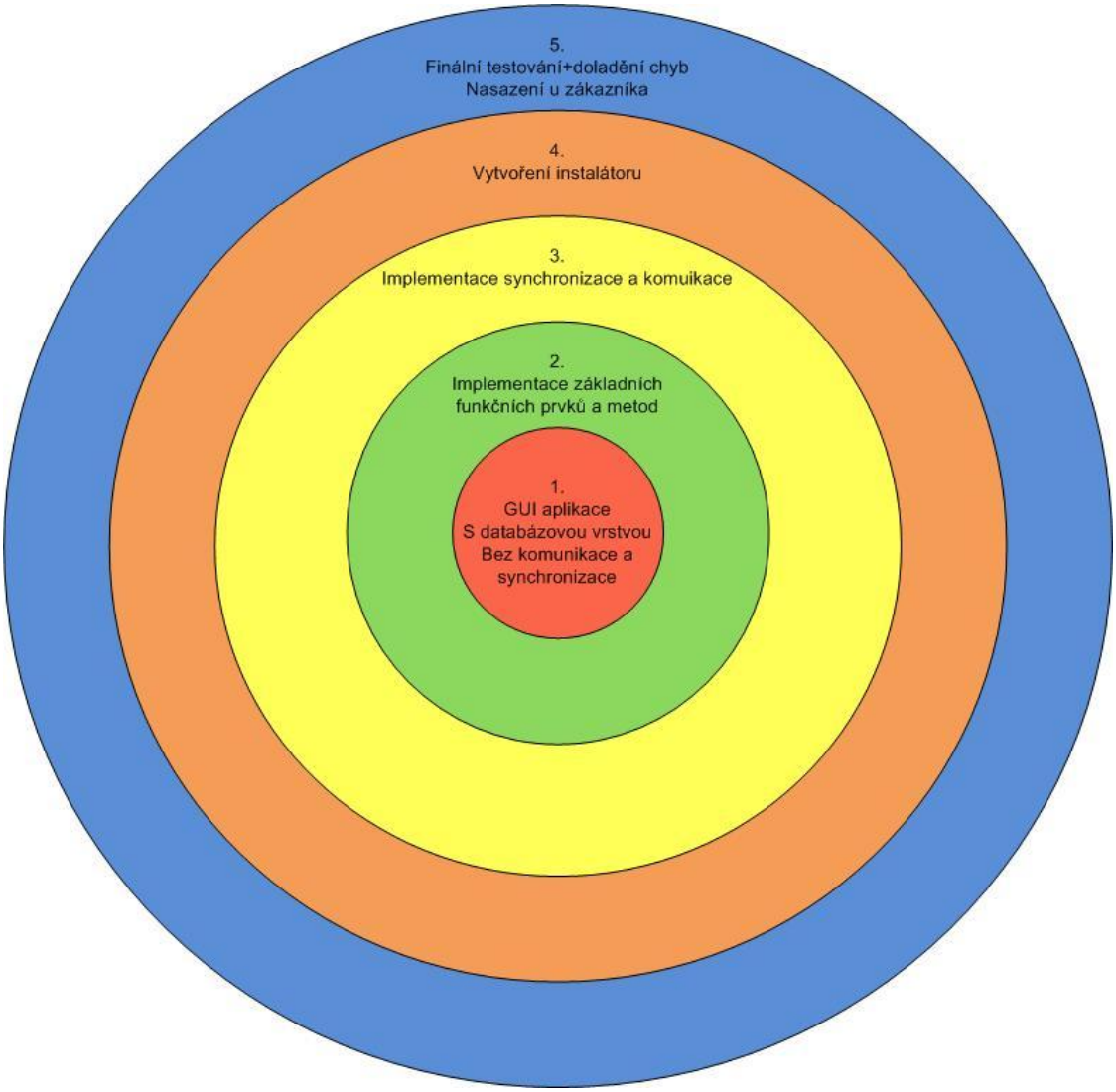
Sekvenční diagram zobrazuje stejnou situaci jako aktivity diagram, ovšem z pohledu spolupráce jednotlivých částí systému v závislosti na čase. V případě přidání nového objektu spolupracují objekty „Uživatel“, „OknoAplikace“, „PrazdneOknoAplikace“ a „DatabazovyServer“. Uživatel v tomto případě odešle požadavek na přidání nového objektu a čeká na odpověď systému (zde na zobrazení prázdného okna pro přidání objektu), poté okno aplikace vyšle příkaz na otevření prázdného okna. Po jeho otevření uživatel zadá data objektu a vyšle požadavek na uložení (opět čeká na odezvu systému), okno nového objektu uloží data do databáze a poté uživatele informuje o uložení objektu do databáze (viz. obr. 5.19).

6 Časový plán vývoje informačního systému

Časový plán byl sestaven s ohledem na nutnost dokončení systému před odevzdáním diplomové práce, počítá se však s dalšími verzemi systému.
Systém je vyvíjen metodou spirály, což znamená, že v každé iteraci probíhá nabalování nových funkcí systému na funkce, které byly implementovány v předchozí iteraci. Vývoj je také veden agilně, což znamená, že v případě potřeby nebude prodlužována doba vývoje, ale bude přizpůsobována funkčnost systému.

Jak je vidět na obrázku 8.1, vývoj probíhá v pěti iteracích:

- 1. Iterace – od 1. do 15. 10 2008 – zahrnuje implementaci GUI aplikace (veškeré obrazovky rozhraní s metodami událostí), včetně databázové vrstvy aplikace. Neobsahuje komunikaci a synchronizaci se zařízením.
- 2. Iterace – 16. 10. – 30. 11. 2008 – v této iteraci probíhá implementace základních funkčních prvků a metod
- 3. Iterace – 1. 12. 2008 – 15. 1. 2009 – v této iteraci je implementována synchronizace a komunikace mobilního zařízení s desktopovým klientem.
- 4. Iterace – 16. 1. – 28. 2. 2009 – v této době proběhne vytvoření instalátoru aplikace
- 5. Iterace - do 30. 4. 2009 – tato iterace zahrnuje konečné odstranění chyb v aplikaci, testování a nasazení systému u zákazníka.



6.1 Zobrazení iterací vývoje systému

7 Návrh systému

Tato kapitola popisuje návrh architektury vyvíjeného systému. Základem architektury je návrhový vzor MVC (Model – View - Control), který umožňuje snadnou modifikaci jednotlivých částí systému a to díky tomu, že jsou na sobě tyto části závislé jen minimálně. Celý systém se skládá ze 4 částí, z databázového serveru, PC aplikace, mobilní aplikace a synchronizačního rozhraní. My se v návrhu zabýváme hlavně architekturou celého systému, návrhem databáze, datového modelu a vzhledu uživatelského rozhraní. Synchronizační rozhraní bylo blíže popsáno v systémové specifikaci. Při návrhu systému nesmíme zapomínat na to, že je úzce spjat a souvisí s analýzou, tyto části se prolínají, proto zde zmiňujeme jen to, co v části „Analýza systému“ zmíněno nebylo a co je pro celkový návrh systému před zahájením implementace potřebné.

7.1 Co jsou návrhové vzory

Návrhové vzory (design patterns) se používají u často se vyskytujících úloh, jako doporučené postupy jejich řešení. Do návrhových vzorů se dosazují třídy, rozhraní a objekty. Výhoda použití návrhových vzorů je v tom, že je výrazně snížena pravděpodobnost výskytu chyb a ulehčení budoucí práce, jelikož návrhové vzory počítají s typickými rozšířeními. Budoucí úpravy tedy budou méně pracné. Dále také usnadňují komunikaci ve skupinách, není totiž potřeba vysvětlovat základní principy, zná – li celá skupina princip návrhových vzorů. Při zápisu návrhových vzorů se používá zakreslení vztahů a postupů řešení pomocí UML diagramů doplněných o slovní popis. Za základní návrhové vzory je považováno 23 návrhových vzorů, které jsou publikovány v katalogu vzorů s názvem GoF.

Základní návrhové vzory publikované v knize GoF:

- Tvořivé vzory (Creational patterns)
  - Vzor tovární metoda (Factory Method Pattern)
  - Vzor abstraktní tovární metoda (Abstract Factory Method Pattern)
  - Vzor Stavitel (Builder Pattern)
  - Vzor prototyp (Prototype Pattern)
  - Vzor jedináček (Singleton Pattern)
- Strukturované vzory (Structural patterns)
  - Vzor adaptér (Adapter Pattern)
  - Vzor most (Bridge Pattern)
  - Vzor strom (Composite Pattern)
  - Vzor dekorátor (Decorator Pattern)
  - Vzor fasáda (Facade Pattern)
  - Vzor Muší váha (Flyweight Pattern)
  - Vzor zástupce (Proxy Pattern)
- Vzory chování (Behavioral patterns)

- Vzor řetěz odpovědnosti (Chain of Responsibility Pattern)
- Vzor příkaz (Command Pattern)
- Vzor interpret (Interpreter Pattern)
- Vzor iterátor (Iterator Pattern)
- Vzor prostředník (Mediator Pattern)
- Vzor pamětník (Memento Pattern)
- Vzor pozorovatel (Observer Pattern)
- Vzor stav (State Pattern)
- Vzor strategie (Strategy Pattern)
- Vzor šablona (Template Pattern)
- Vzor návštěvník (Visitor Pattern) [14].

7.2 Návrhový vzor MVC a důvod jeho použití

Pro vyvíjený systém byl vybrán návrhový vzor, který sice nepatří mezi výše zmiňované základní vzory, to však neznamená, že je méně výhodný. Tento vzor se nazývá Model-View-Controler (Model-Pohled-Ovládání). Týká se spíše architektury systému. Základním principem je oddělení části programu, která se stará o předzpracování příkazů uživatele od logické části zpracovávající příkazy a od části zobrazující výsledky. Jinak řečeno odděluje uživatelské rozhraní od logiky a datového modelu. Každou z těchto částí má nestarosti samostatný objekt nebo skupina objektů. Výhodou tohoto přístupu je, že lze tímto zajistit, aby aplikace reagovala na vstup nezávisle na tom, jakým způsobem jsou vstupy zadány (klávesnice, myš, atd..), také díky nezávislosti všech tří modulů tohoto vzoru budou při zásahu (změně) do jedné části ovlivněny zbylé dvě jen minimálně.

Vstupy – zahrnují reakce na události vyvolané prostřednictvím ovládacích prvků (např. tlačítka, radiobuttony, atd.).

Výstupy – neobsahují pouze vykreslování hodnot, ale je možné do této části zahrnou také transformaci získaných výsledků do podoby, ve které budou reprezentovány uživateli. Forma požadovaného výstupu je libovolná (GUI, tiskárna, ukládání do souboru), uživatel také může požadovat pokaždé jinou formu výstupu.

Model – je vnitřní reprezentací zpracovávaných dat a zahrnuje logiku, která tyto data zpracovává. [14]

7.3 Návrh vyvíjeného systému

Návrh architektury systému

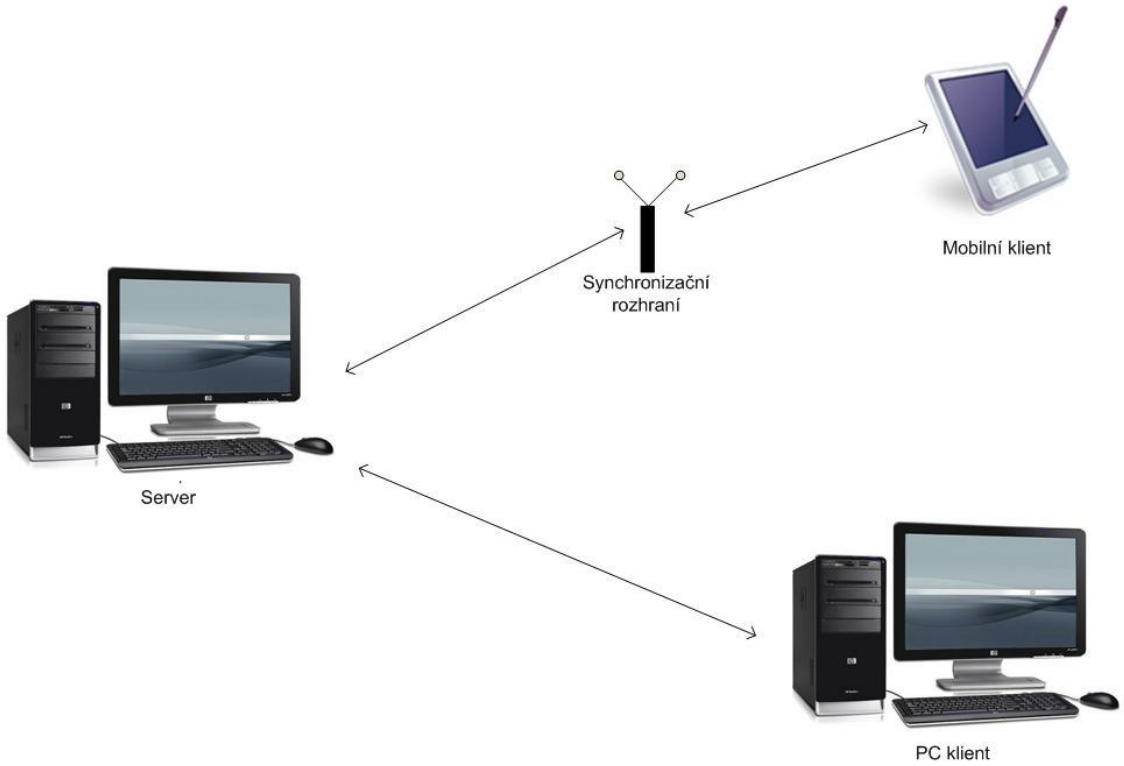
Jak již bylo uvedeno v systémové specifikaci, architektura systému se bude skládat ze čtyř částí a to z: **mobilního klienta**, jenž představuje mobilní zařízení používané zaměstnanci agentury při práci v terénu,

PC klienta, pro správu pacientů a číselníků přímo v místě agentury,

Serverové části, pro uchovávání veškerých dat potřebných k práci,

Synchronizačního rozhraní, sloužícího k synchronizaci dat, mezi jednotlivými částmi systému.

Jak je zobrazeno na obrázku 7.1, PC klient a server budou propojeny stále pomocí síťového kabelu. Také se může jednat o jedno a totéž zařízení, v praxi by však bylo lepší tyto 2 zařízení od sebe oddělit, a to z toho důvodu, že v případě havárie jednoho z nich přijde uživatel buď jen o aplikaci, nebo jen o data, pokud by byly obě části jedním zařízením, přišel by uživatel o obojí. Veškerá data musí být samozřejmě zálohována. Synchronizace mobilního klienta je prováděna přes již zmíněné synchronizační rozhraní. Veškerá synchronizace probíhá offline, z důvodu ochrany choulostivých dat pacientů. Synchronizace bude tedy probíhat pomocí standardu pro lokální bezdrátové sítě WiFi, také by bylo možné provést synchronizaci technologiemi bluetooth nebo USB.



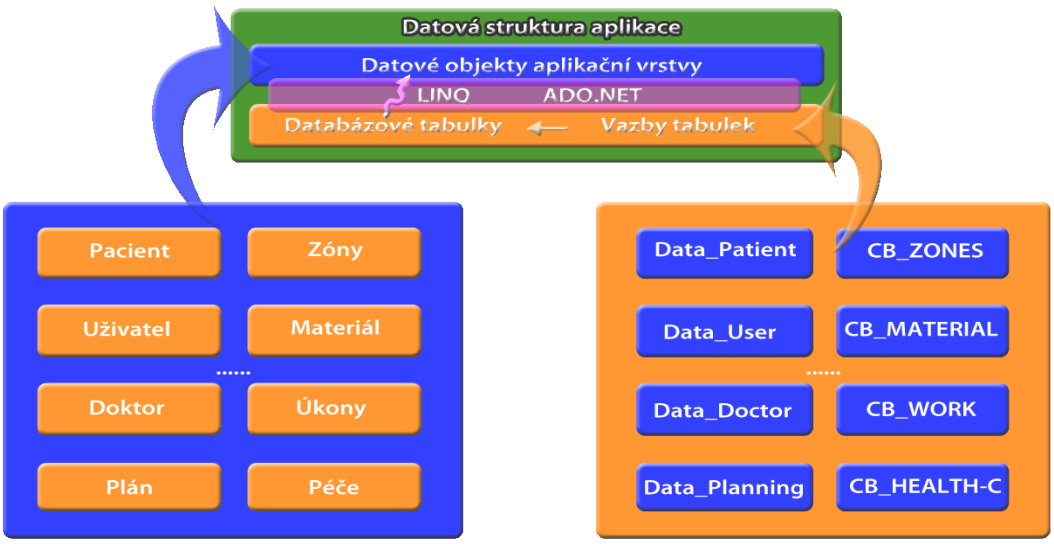
Obrázek 7.1 Architektura návrhového systému

Návrh databáze:

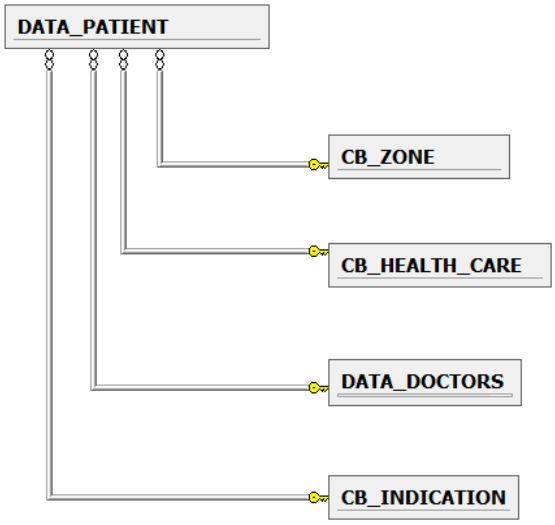
Pro aplikaci byla nejprve navrhována databáze. Vzhled jejich tabulek a forma dat, které budou tabulky obsahovat. Na obrázku 7.2 je vidět datový model navržený pro tuto aplikaci.

Datový model se skládá na jedné straně z datových objektů aplikační vrstvy, tato vrstva je reprezentována objekty v aplikaci pojmenovanými příznačně podle toho, jaká data v aplikaci reprezentují. Například objekt „Pacient“ reprezentuje v aplikaci osobní i zdravotní informace o pacientovi. Na druhé straně se datový model skládá z databázových tabulek, tyto tabulky obsahují uložená veškerá data, která jsou poté reprezentována již zmíněnými datovými objekty. Jeden datový objekt však nemusí reprezentovat pouze jedna databázová tabulka, například datový objekt pacient je fyzicky reprezentován tabulkami Data_Patient, CB_Zone, CB_Health_Care, Data_Doctors a CB_Indication. A to z toho důvodu, že při tvorbě databáze držíme zásady, že data, která se mohou vyskytovat ve více tabulkách, musí být fyzicky

obsažena pouze v jedné tabulce, na kterou se ostatní tabulky, které tyto data také používají, budou pouze odkazovat pomocí primárního klíče (viz obr. 7.3). Díky tomuto opatření nemůže dojít k tomu, že při změně některých dat, budou tato data změněna pouze v jedné tabulce a v ostatních ne. Primární klíč, pomocí kterého se tabulky odkazují na potřebná data, musí být unikátní a neměnný. Propojení mezi datovými objekty a databázovými tabulkami obsahující tyto potřebná data je provedeno technologiemi LINQ pro PC klienta a ADO.NET pro mobilního klienta.



Obrázek 7.2 Datový model



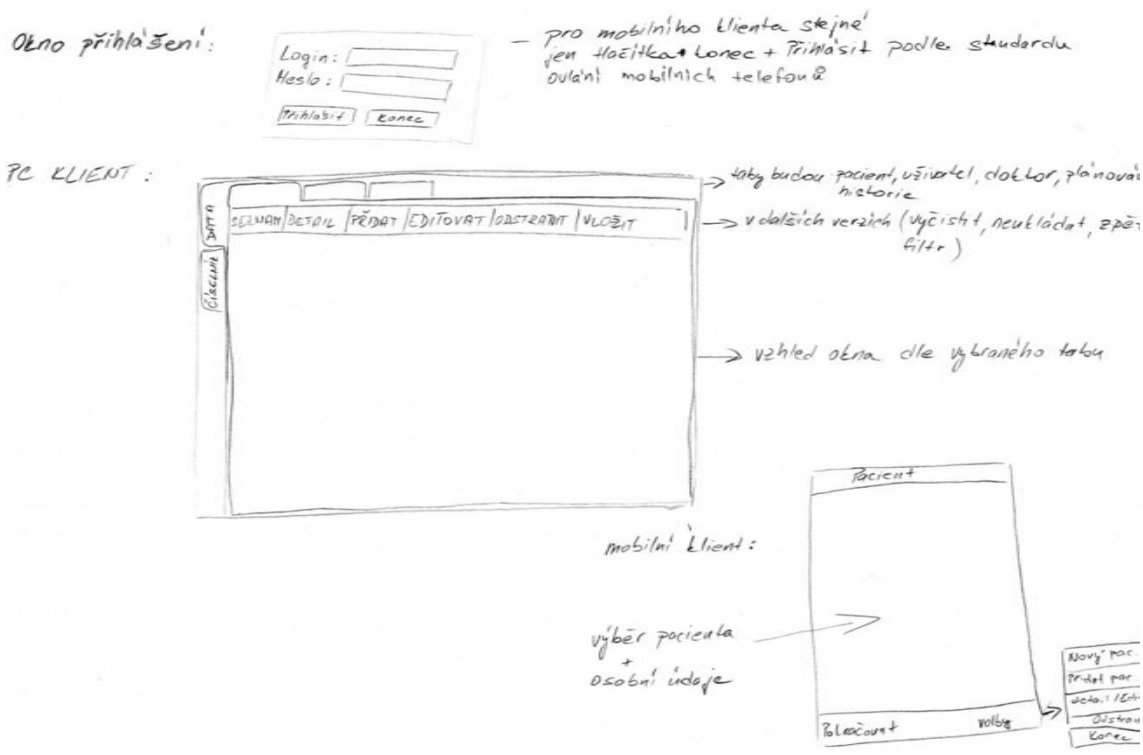
Obrázek 7.3 Relace tabulky DATA_PATIENT

Návrh uživatelského rozhraní

Dále byl navrhován vzhled aplikace. Prvotní ručně kreslený návrh je uveden na obrázku 7.4. Tento návrh obsahuje vzhled okna pro přihlášení, ten má klasický vzhled obsahující pole pro zadání přihlašovacího jména, okno pro zadání hesla a tlačítka *Přihlásit* a *Konec*.

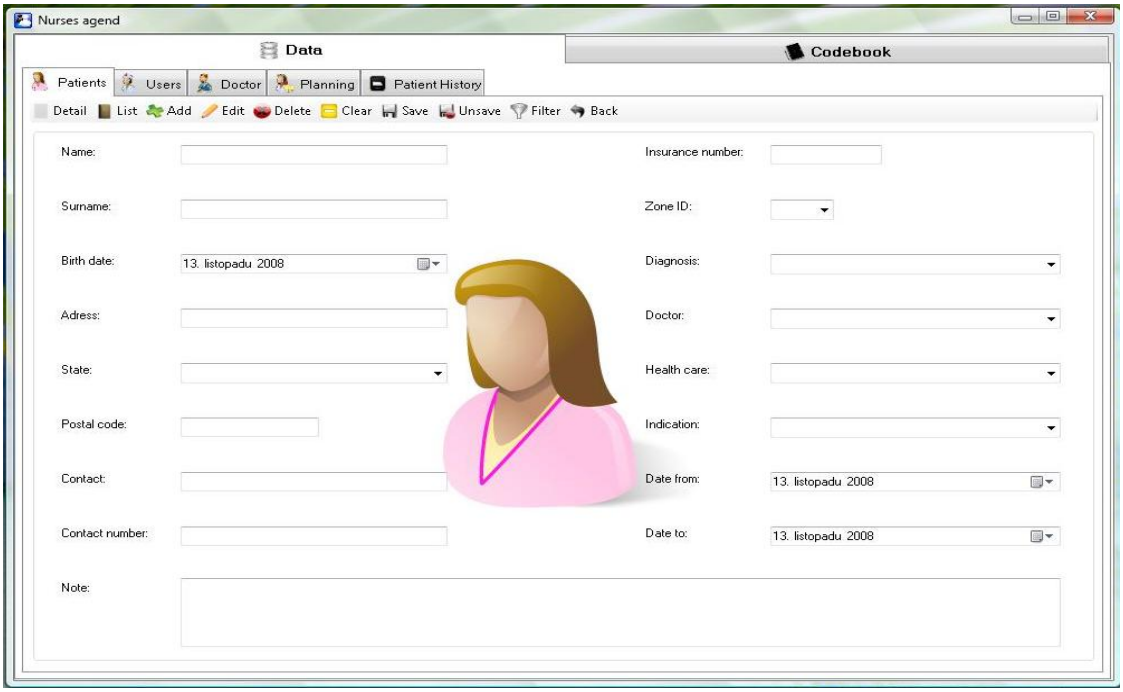
Hlavní okno PC klienta bylo rozděleno do dvou hlavních částí (2 svislé taby vlevo nahoře) a to na část pro práci s daty a část pro práci s číselníky. Každý z těchto dvou svislých tabů dále obsahuje sadu vodorovných tabů, ty reprezentují přímo jednotlivé objekty se kterými je v sadě *Data* nebo *Číselníky* možno pracovat (pro data je to například objekt pacient, Doktor atd. pro číselníky je to objekt Materiál, Zóny atd.). Dále každý tab obsahuje funkční tlačítka pro práci s objekty a to tlačítka pro zobrazení seznamu, detailu objektu, přidání objektu, editace objektu, odstranění objektu a uložení změn. V dalších verzích programu se počítá i s tlačítky pro pokročilou volbu jako například filtr (viz. systémová specifikace).

Posledním navrhovaným oknem bylo okno mobilní aplikace. Toto okno se pro jednoduchost drží klasického vzhledu aplikací pro mobilní zařízení. Ve spodní části obsahuje 2 tlačítka. Vpravo je to tlačítko *Volby* pro například přidání dalšího pacienta nebo editaci stávajícího atd. Druhé tlačítko je tlačítko *Pokračovat*, tímto tlačítkem se uživatel dostane k následujícímu oknu práce (například po výběru pacienta, se po stisku tlačítka *Pokračovat* zobrazí seznam úkolů pro daného pacienta).



Obrázek 7.4 Původní ruční návrh okna aplikace

Následující obrázek 7.5 ukazuje nynější vzhled hlavního okna aplikace. Oproti prvotnímu návrhu nejsou 2 hlavní taby ve svislé poloze, ale také ve vodorovné poloze, stejně jako následující objekty. K této změně došlo po domluvě s vývojářem aplikace a to z důvodu větší přehlednosti pro uživatele.



Obrázek 7.5 Nynější vzhled okna aplikace

8 Testování systému

Vývoj jakéhokoliv softwaru nutně doprovází také jeho testování, ovšem jeho výskyt se liší projekt od projektu a také v závislosti na firmě, která systém vyvíjí. Každá firma totiž dává přednost jinému typu modelu vývoje a od tohoto se také odvíjí, kdy se vyvíjený systém začíná testovat. Této problematice byla věnována kapitola 4, této práce. Testování má za úkol vyhledávání chyb a nesprávné funkčnosti vyvíjeného systému.

8.1 Co je chyba

Chceme – li testovat software a tím odhalovat jeho chyby, musíme vědět, co to vlastně chyba je. Chyby totiž mohou být jak téměř bezvýznamné, dojde – li k takové chybě, nemusí ji uživatel ani postřehnout. U takovýchto chyb může také dojít k tomu, že nebude naprosto jisté, že se vůbec o chybu jedná. Na druhou stranu existují chyby, jejichž výskyt má třeba i tragické následky.

V případě chyby týkající se selhání softwaru se můžeme setkat s pojmy:
Závada, selhání, vada – toto označení značí ve většině případů nějaký závažný problém, který by mohl být třeba i nebezpečný. Značí, že systém z důvodu nějaké chyby selhal.
Anomálie, odchylka, událost - tyto pojmy mají menší důležitost, než předchozí, značí spíše nějaké neočekávanou činnost systému nebo neúmyslnou chybu, nejedná se o závažné selhání systému.
Problém, omyl, chyba – nejčastěji používané chyby.

Pokud chceme přesně určit, co v softwaru je a co není chyba, musíme pojem softwarová chyba definovat. Za chybu v softwaru tedy považujeme problém, který splňuje některou z následujících definicí:

- Software nesplňuje nějaké chování, které by podle specifikace splňovat měl (nedělá něco, co má zadáno ve specifikaci).
- Software obsahuje chování, které podle specifikace obsahovat nemá (dělá něco, co ve specifikaci nemá napsáno).
- Softwaru chybí nějaká funkce, která sice není napsaná ve specifikaci, ale napsaná by v ní měla být.
- Se softwarem se špatně pracuje, je pomalý nebo nesrozumitelný.

Náklady na opravu chyby se s prodlužující se dobou, kdy je chyba odhalena, logaritmicky zvyšují, proto je velice důležité chyby v softwaru odhalit včas, nejlépe hned na samotném začátku vývoje systému, tudíž již při psaní specifikace a to tak, že bude specifikace důkladně probrána všemi členy týmu. [3]

8.2 Testování bílé a černé skříňky

Způsob testování je rozdělen do dvou hlavních typů postupů a to je testování černé skříňky a testování bílé skříňky. Jak napovídá již název, první z nich, testování černé skříňky znamená, že je systém testován tak, že jsou zadány nějaké vstupy systému a aniž bychom tušili, jakým způsobem systém uvnitř pracuje a data na vstupu zpracovává, očekáváme výstupy odpovídající tomu, co má systém dělat. Uživatel (tester) netuší, jak k těmto výsledkům systém došel, jen je pozoruje a určuje, zda odpovídají jeho očekávání nebo ne. Tomuto je samozřejmě přizpůsoben i způsob výběru testovacích scénářů, které jsou vybírány pouze podle systémové specifikace. Je nutné sledovat také reakce na mezní nebo špatná data zadaná systému. Příkladem tohoto typu testování je například testování specifikace nebo testování GUI.
Testování bílé skříňky je pravým opakem černé skříňky, tudíž tester má při testování přístup ke zdrojovému kódu. Pohlíží – li tester i na programový kód, může snadněji odhalit hodnoty vstupů, které budou způsobovat problémy. Testování bílé skříňky má prověřit veškeré možné způsoby průchodu programem. Používá se většinou při testování jednotlivých částí (komponent) aplikace. [3]

8.3 Statické a dynamické testování

Hlavním rozdílem ve statickém a dynamickém testování je, že při statickém testování program není prováděn. Statické testování má za úkol nalézt chyby, aniž by byl testovaný systém spuštěn, je tedy pouze zkoumán a revidován. Například verifikace programu, která kontroluje správnost návrhu a jeho implementaci, inspekce kódu, což je pozorné čtení programu, analýza struktury atp.
Při dynamickém testování je software spuštěn, pracuje se s ním a je za běhu kontrolován. Činnost systému může být také pouze simulována. V případě, že proběhne statické testování úspěšně, musí systém a každá jeho část projít úspěšně i dynamickým testováním. To probíhá tak, že se vytvoří testovací prostředí, poté testovací scénáře a nakonec jsou testy provedeny a zpracovány do přehledných výsledků.
Výše zmíněné testování specifikace se řadí do statického testování černé skříňky, mezi dynamické testování černé skříňky se říká také testy chování, jedná se o dynamické testování z důvodu, že testovaný systém běží a černá skříňka proto, že tester neví, jak systém pracuje (například testy splněním a testy selháním). [3]

8.4 Testování systémové specifikace vyvíjeného systému

Tato kapitola testování má za úkol zjistit, zda vyvíjená aplikace vyhovuje požadavkům zadaným agenturou a sepsaným v systémové specifikaci. I funkční systém může být totiž nevyhovující, pokud nejspou splněny všechny požadavky, klafené na systém. Tyto nesrovnalosti by totiž mohly ovlivnit práci se systémem. Například řekněmě, že by v aplikaci bylo správně dle specifikace implementováno přidání nového pacienta, ale chyběla by jeho editace, potom by v případě potřeby změny některého z jeho údaju byl uživatel systému nucen nejdříve přidat nový záznam s provedenými změnami v datech a poté umazat starý záznam. S tímto by však

mohly souviset další komplikace a to například v případě, že by uživatel zapomněl původní záznam umazat, pak by vznikaly kolize při vytvoření plánu. Uživatel by chtěl naplánovat úkon pacientovi, ten by však měl v databázi 2 záznamy, které by se lišily třeba jen v jedné položce, uživatel by však nemusel vědět, který z těchto dvou záznamů je ten správný a aplikace by měla problém určit, kterému z těchto dvou záznamů přiřadit plán, jelikož se plán přiřazuje pouze podle jména pacienta. Takových problémů by mohlo nastat mnoho a proto je potřeba otestovat, jestli opravdu specifikaci systém odpovídá.

Při testování specifikace byly zjištěny tyto nesrovnalosti:

- Oproti původní specifikaci bude v budoucnu přidána další komponenta systému – systém se tedy nebude skládat ze 4 komponent, ale z pěti, pátá komponenta je webová služba, přes kterou bude realizován přístup k databázi (systémová specifikace kapitola 2.1 kontext systému). V důsledku toho bylo provedeno testovací napojení aplikace na webovou službu a zátěžové testy – viz kapitola 8.4.

- Uživatelský manuál mobilního klienta bude realizován oproti specifikaci až po otestování klienta zdravotními sestrami (Systémová specifikace kapitola 2.6).

- Odstraňování jakýchkoliv položek číselníků je z důvodu kolizí možné jen v případě, že tato položka není použita v žádném plánu, nebo například materiál nesmí být přiřazen ani žádné práci.

- V této verzi nejsou implementovány práce se skupinami a dietami – jak již bylo zmíněno, aplikace je vyvíjena agilně, takže funkčnost je přizpůsobována času. Tato funkce zatím není nezbytná, a proto bude implementována v dalších verzích.

- Specifikace udává, že odezva na požadavek nesmí být delší než 5 s (systémová specifikace kapitola 5.1). To by v případě simultánního přístupu 1000 uživatelů k serveru uloženém na PC s procesorem AMD 850MHz a pamětí 512MB nebylo splněno, v tomto případě je odezva dlouhá až 7 sekund a s přibývajícím počtem záznamů v databázi by se tato doba dále zvyšovala, tento typ serveru byl však jen pro srovnání, specifikaci by nevyhovoval ani parametrem své paměti, ta musí být alespoň 2 GB. PC, který je reálně použit jako server, specifikaci vyhovuje ve všech bodech, pro simultánní přístup tisíce uživatelů dosahuje odezvy dlouhé maximálně 2s, výkonnostním požadavkům tedy bude vyhovovat.

V ostatních bodech aplikace specifikaci vyhovuje.

8.5 Testování uživatelského rozhraní

Při testování uživatelského rozhraní se zjišťuje, zda systém při použití různými způsoby pracuje očekávaným způsobem, to znamená, zda se dopracujeme k očekávanému výsledku. Očekávaný výsledek však znamená výsledek popsaný ve specifikaci, ne výsledek, který v danou chvíli napadne uživatele nebo testera. Proto jsou testy, které budou při testování uživatelského rozhraní předem plánovány a zapisovány. Při plánování testů, k teré budou prováděny, je tedy

teste opět odkázán na specifikaci systému a podle ní určuje, co bude testovat a jaké výsledky bude poté očekávat od systému.

Účelem testování uživatelského systému, je však také zjistit, jestli je práce se systémem dostatečně uživatelky příjemná a intuitivní. Což znamená, že i při prvním zhlédnutí by uživatel, který se systémem není dostatečně seznámen, neměl mít žádné zásadnější problémy s ovládáním systému. Toto testování závidí však výhradně na osobním názoru a zkušenostech testera.
V případě tohoto systému jsem po práci s ním usoudila, že rozdělení aplikace do záložek a následně podzáložek, je velice přehledná a uživatel by s ní neměl mít problémy, vidí totiž přesně ve které záložce a podzáložce se nacházía nemusí složitě hledat, jakým způsobem by se z této kombinace přepl do jiné části aplikace, rozhodne – li se upravit nějaký číselník, přepne se jednoduše do záložky číselníků, kde má opět přehledný seznam objektů v podobě podzáložek, se kterými nyní může pracovat. Následná práce s daty je taky velice přehledná, tlačítka k tomu určená jsou ve všech záložkách stejné, uživatel si tedy rychle zvykne a to, které tlačítko má použít a kde jej najde. Další výhodou uživatelské aplikace je to, že se uživateli nemůže stát, že by se s daty snažil provést nějakou právě nevhodnou manipulaci, tlačítka, které totiž v danou chvíli není vhodné použít, jsou pro použití zakázána, uživatel tedy na první pohled vidí, že je použít nemůže.

Uživatelské rozhraní PC klienta je testováno pro typ uživatele *Administrátor* a to proto, že tato role má přístupné veškeré funkce aplikace, takže takto otestujeme kompletní potřebnou funkcionalitu aplikace. Při testování budou popsány scénáře použití aplikace do přehledné tabulky zahrnující výsledek testu. Pro otestování správné funkce znepřístupnění některých funkcí rolím, které nemají tyto funkce povoleny, jsou vytvořeny další tabulky, ve kterých jsou napsány typy funkcí, které mají být znepřístupněny dané roli a výsledek testu (tzn., zda byla funkce opravdu zakázána).

Scénáře způsobu možného použití aplikace:

Typ role: Administrator		
Název scénáře	**Popis**	**Výsledek testu**
Přidání uživatele	Kliknout na tlačítko *Přidat* z okna seznamu uživatelů nebo z detailu uživatele; vyplnit veškeré povinné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam uživatel, nebo detail přidaného uživatele).	OK
Editace uživatele	Kliknout na tlačítko *Editovat* z okna seznamu uživatelů nebo z detailu uživatele; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění uživatele	Kliknout na tlačítko *Odstranit* z okna seznamu uživatelů nebo z detailu uživatele; požadovaný uživatel bude přidáním atributu odstraněn a aplikace se vrátí do výchozího okna (již bez odstraněného uživatele).	OK

Přidání doktora	Kliknout na tlačítko *Přidat* z okna seznamu doktorů nebo z detailu doktora; vyplnit veškeré povinné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam lékařů, nebo detail přidaného lékaře).	OK
Editace doktora	Kliknout na tlačítko *Editovat* z okna seznamu lékařů nebo z detailu lékaře; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění doktora	Kliknout na tlačítko *Odstranit* z okna seznamu doktorů nebo z detailu doktora; uživatel bude požádán o potvrzení smazání; požadovaný lékař bude přiřazením atributu odstraněn a aplikace se vrátí do výchozího okna (již bez odstraněného lékaře).	OK
Přidání pacienta	Kliknout na tlačítko *Přidat* z okna seznamu pacientů nebo z detailu pacienta; vyplnit veškeré povinné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam pacientů, nebo detail přidaného pacienta).	OK
Editace pacienta	Kliknout na tlačítko *Editovat* z okna seznamu pacientů nebo z detailu pacienta; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění pacienta	Kliknout na tlačítko *Odstranit* z okna seznamu pacientů nebo z detailu pacienta; uživatel bude požádán o potvrzení smazání; požadovaný pacient bude přidáním atributu odstraněn a aplikace se vrátí do výchozího okna (již bez odstraněného pacienta).	OK
Vytvoření plánu	Kliknout na tlačítko *Přidat* v okně seznamu plánů nebo z detailu plánu; vyplnit veškeré potřebné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam plánů, nebo detail přidaného plánu).	OK
Editace plánu	Kliknout na tlačítko *Editovat* z okna seznamu plánů nebo z detailu plánu; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění plánu	Kliknout na tlačítko *Odstranit* z okna seznamu plánů nebo z detailu plánu; uživatel bude požádán o potvrzení smazání; požadovaný plán bude odstraněn a aplikace se vrátí do výchozího okna (již bez odstraněného pacienta).	OK
Editace historie	Kliknout na tlačítko *Editovat* v okně seznamu historie nebo z detailu vybrané historie; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Tisk historie	Kliknout na tlačítko *Tisk* v okně detailu dané historie; v popup okně zadat nastavení tisku; potvrdit tisk; historie se vytiskne a okno zůstane v detailu tisknuté historie.	OK
Přidání zóny	Kliknout na tlačítko *Přidat* v okně seznamu zón nebo z detailu zóny; vyplnit veškeré potřebné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam zón, nebo detail přidané zóny).	OK

Editace zóny	Kliknout na tlačítko *Editovat* v okně seznamu zón nebo z detailu vybrané zóny; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění zóny	Kliknout na tlačítko *Odstranit* z okna seznamu zón nebo z detailu zóny; uživatel bude požádán o potvrzení smazání; požadovaná zóna bude odstraněna a aplikace se vrátí do výchozího okna (již bez odstraněné zóny).	OK
Přidání materiálu	Kliknout na tlačítko *Přidat* v okně seznamu materiálů nebo z detailu materiálu; vyplnit veškeré potřebné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam materiálů, nebo detail přidaného materiálu).	OK
Editace materiálu	Kliknout na tlačítko *Editovat* v okně seznamu materiálů nebo z detailu vybraného materiálu; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění materiálu	Kliknout na tlačítko *Odstranit* z okna seznamu materiálů nebo z detailu materiálu; uživatel bude požádán o potvrzení smazání; požadovaný materiál bude odstraněn a aplikace se vrátí do výchozího okna (již bez odstraněného materiálu).	OK
Editace role	Kliknout na tlačítko *Editovat* v okně seznamu rolí nebo z detailu vybrané role; editovat požadované položky či práva; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Přidání úkolu	Kliknout na tlačítko *Přidat* v okně seznamu úkolů nebo z detailu úkolu; vyplnit veškeré potřebné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam úkolů, nebo detail přidaného úkolu).	OK
Editace úkolu	Kliknout na tlačítko *Editovat* v okně seznamu úkolů nebo z detailu vybraného úkolu; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění úkolu	Kliknout na tlačítko *Odstranit* z okna seznamu úkolů nebo z detailu úkolu; uživatel bude požádán o potvrzení smazání; požadovaný úkol bude odstraněn a aplikace se vrátí do výchozího okna (již bez odstraněného úkolu).	OK
Přidání diagnózy	Kliknout na tlačítko *Přidat* v okně seznamu diagnóz nebo z detailu diagnózy; vyplnit veškeré potřebné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam diagnóz, nebo detail přidané diagnózy).	OK
Editace diagnózy	Kliknout na tlačítko *Editovat* v okně seznamu diagnóz nebo z detailu vybrané diagnózy; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění diagnózy	Kliknout na tlačítko *Odstranit* z okna seznamu diagnóz nebo z detailu diagnózy; uživatel bude požádán o	OK

	potvrzení smazání; požadovaná diagnóza bude odstraněna a aplikace se vrátí do výchozího okna (již bez odstraněné diagnózy).	
Přidání zdravotní péče	Kliknout na tlačítko *Přidat* v okně seznamu zdravotních péčí nebo z detailu zdravotní péče; vyplnit veškeré potřebné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam zdravotních péčí, nebo detail přidané zdravotní péče).	OK
Editace zdravotní péče	Kliknout na tlačítko *Editovat* v okně seznamu zdravotních péčí nebo z detailu vybrané zdravotní péče; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění zdravotní péče	Kliknout na tlačítko *Odstranit* v okně seznamu zdravotních péčí nebo z detailu zdravotní péče; uživatel bude požádán o potvrzení smazání; požadovaná zdravotní péče bude odstraněna a aplikace se vrátí do výchozího okna (již bez odstraněné zdravotní péče).	OK
Přidání indikace	Kliknout na tlačítko *Přidat* v okně seznamu indikací nebo z detailu indikace; vyplnit veškeré potřebné údaje; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna (obnovený seznam indikací, nebo detail přidané indikace).	OK
Editace indikace	Kliknout na tlačítko *Editovat* v okně seznamu indikací nebo z detailu vybrané indikace; editovat požadované položky; kliknout na tlačítko *Uložit;* aplikace se vrátí do výchozího okna.	OK
Odstranění indikace	Kliknout na tlačítko *Odstranit* v okně seznamu indikací nebo z detailu indikace; uživatel bude požádán o potvrzení smazání; požadovaná indikace bude odstraněna a aplikace se vrátí do výchozího okna (již bez odstraněné indikace).	OK

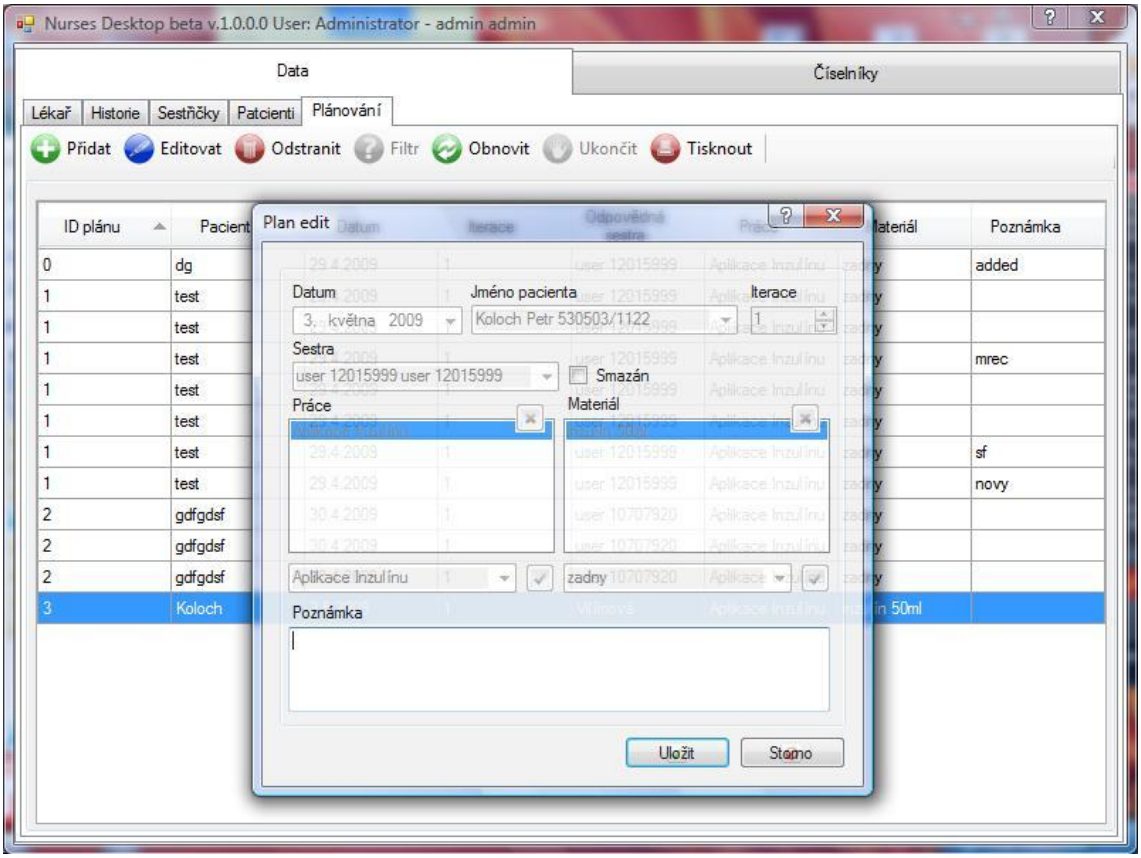
Tabulka 1 Přehled scénářů použití aplikace

Výsledky funkčnosti znepřístupnění nepovolených funkcí:

Název zakázané funkce	Výsledek testu
Typ role: Hlavní sestra	
Práce s testy a statistikami	Funkce v této verzi není implementována
Typ role: Zdravotní sestra	
Přidat uživatele	OK
Editovat uživatele	OK
Odstranit uživatele	OK
Přidat doktora	OK
Editovat doktora	OK
Odstranit doktora	OK
Editovat historii	OK
Tisknout historii	OK
Práce s testy a statistikami	Funkce v této verzi není implementována

Tabulka 2 Přehled nepovolených funkcí dle rolí

Při testování uživatelského rozhraní bylo zjištěno, že nynější způsob otevírání oken například pro vytvoření nového uživatele nebo editaci v hlavním okně aplikace (v tabu vybraného objektu) může, nedrží – li se uživatel uživatelské dokumentace k aplikaci, to znamená pracuje s ní tak, jak jej zrovna napadne (klasický způsob práce všech uživatelů), způsobovat kolize. Proto byla aplikace po dohodě předělána a tyto okna nejsou zobrazována v hlavním okně aplikace, ale jako PopUp okno (viz. obrázek 8.1).



Obrázek 8.1 Vzhled nové aplikace

Dále je nutné zařadit do časového plánu další 2 iterace vývoje systému a to následující:
6. iterace – optimalizace systému – načtení údajů z databáze probíhá v některých případech bez problémů, jindy toto trvá přibližně minutu. Tato iterace je plánována na období 1. Května – 30. července
7. předání beta verze aplikace – Beta verze bude agentuře po domluvě s majitelkou agentury předána v průběhu srpna roku 2009.

8.6 Zátěžové testování přístupu k datům přes webovou službu

Zátěžové testování udává, jak se systém chová v případě, přistupuje – li k jeho datům větší množství uživatel současně. Pro testování ovlivnění délky časové odezvy počtem simultánních přístupů k databázi byla vyvinuta aplikace v jazyce C# a vývojovém prostředí Microsoft Visual Studio 2008.

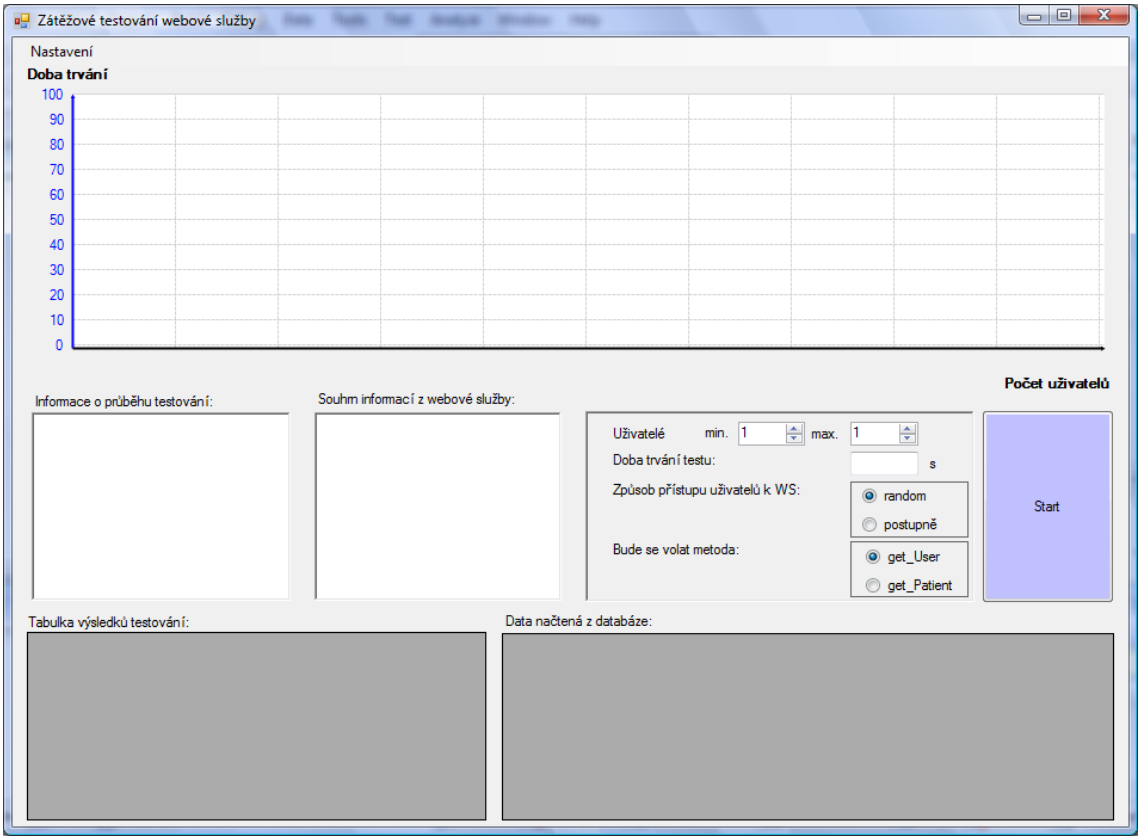
Software má za úkol simulovat přístup libovolného počtu uživatelů současně, k metodám implementovaným na webové službě a měřit dobu odezvy webové služby na požadavek v závislosti na počtu uživatelů, kteří k službě v danou chvíli přistupují. Aplikace měří dobu trvání jednoho zavolání dané metody, dobu trvání volání metody při zadaném počtu simultánních přístupů uživatelem, celkovou dobu odezvy webové služby daným počtem uživatelů a to včetně doby dopravy požadavku na webovou službu, sestavení kolekce požadovaných dat z databáze a cesty této kolekce zpět z webové služby k aplikaci a dobu trvání vykonání kompletního testovacího požadavku od stisknutí tlačítka start po ukončení testu.

Aplikace umožňuje:

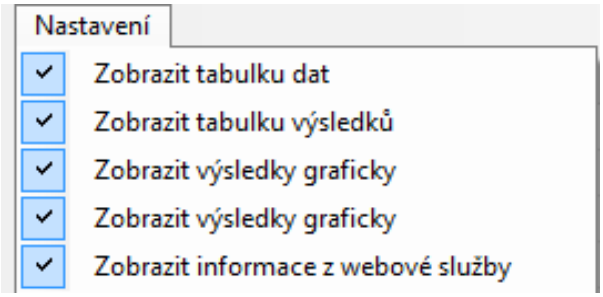
- zadat maximální počet přistupujících uživatelů,
- vybrat metodu, kterou budeme volat,
- zadat způsob jakým bude přistupována ke službě (random, postupně),
- zadat dobu, po kterou má být test prováděn (platí jen pro random přístup).

Vzhled uživatelského rozhraní a ovládání aplikace

Okno aplikace (obrázek 8.2) obsahuje jednu položku menu s názvem „Nastavení“, ta uživateli umožňuje nastavit vzhled aplikačního okna, nebo lépe skrýt (zobrazit) některé položky aplikačního okna a to „Bar graf“, „Informace o průběhu testování“, „Souhrn informací z webové služby“, „Tabulka výsledků testování“ a „Tabulka dat z databáze“. Tyto položky může uživatel skrýt/zobrazit odškrtnutím/zaškrtnutím dané položky v menu „Nastavení“. Detailní pohled na menu „Nastavení“ je zobrazen na obrázku 2. Dále okno obsahuje Bar graf, pro grafické zobrazení výsledku testu, dvě textová pole, jedno zobrazuje informace o právě proběhlém testování a druhé zobrazuje informace získané z webové služby o průběhu volané metody, také je do něj zapsán údaj o režii webové služby na době trvání testování, což je doba potřebná k navázání spojení s webovou službou a její kompilace, tento údaj je totiž schopen výsledky testování zkreslit. Vedle těchto dvou polí je sekce s ovládacími prvky pro nastavení požadovaného testu (viz. obrázek 8.4). Ve spodní části aplikačního okna jsou pak 2 tabulky, jedna obsahuje data načtená z databázového serveru a druhá informace o ukončeném testu.

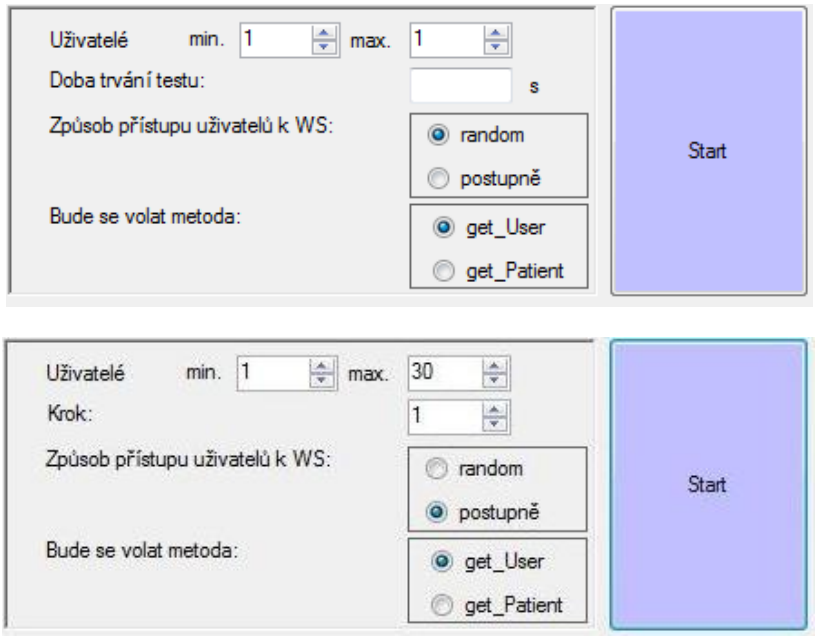


Obrázek 8.2 Vzhled testovací aplikace



Obrázek 8.3 Detail menu "Nastavení"

Na obrázku 8.4 je zobrazen detail sekce ovládání. Ta obsahuje prvek pro výběr metody, která bude volána (getUser nebo getPatient), prvek pro výběr způsobu, jakým bude generován počet uživatel přistupujících k webové službě (random nebo postupně), textové pole pro zadání doby, po kterou má být test prováděn (v případě random přístupu), prvek pro zadání velikosti kroku při postupném generování uživatelů a prvky pro zadání minimálního a maximálního počtu uživatelů, kteří mají k webové službě současně přistupovat.

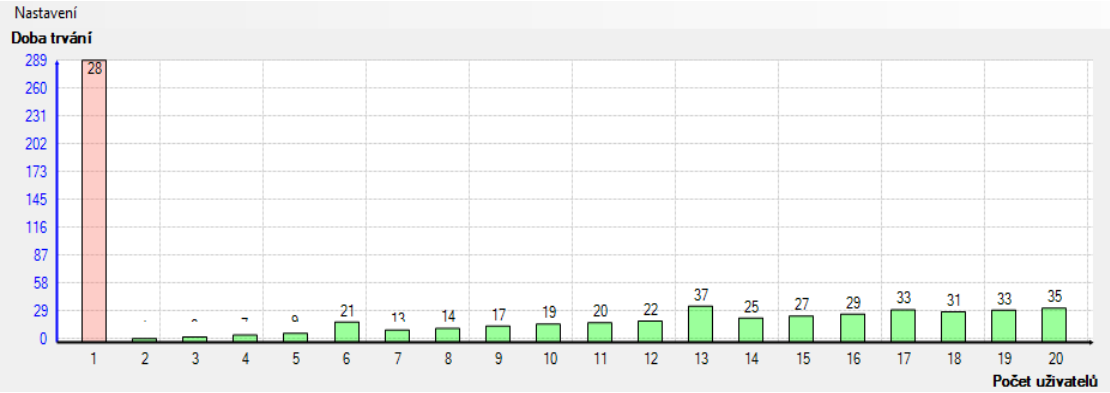


Obrázek 8.4 Detail sekce ovládacích prvků

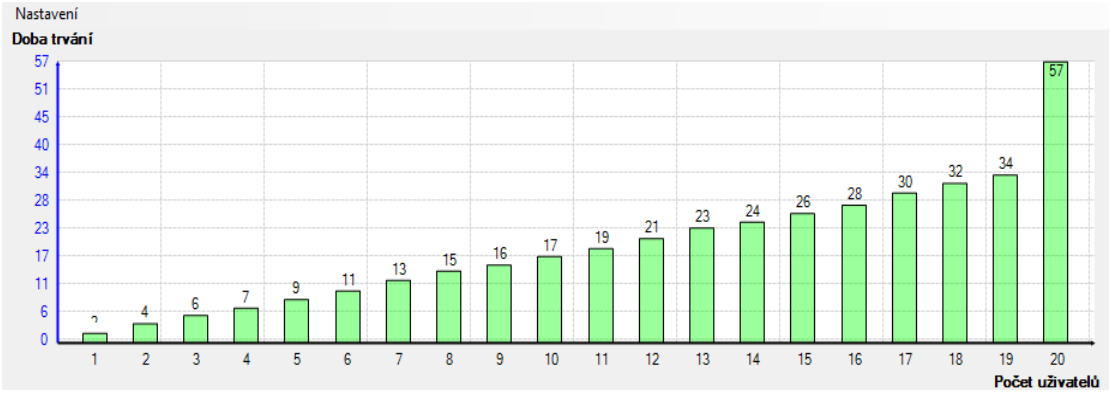
Způsob práce programu:

V případě zvolení postupného generování uživatelů, bude k metodě najednou přistupovat nejprve minimální zadaný počet uživatel a v každém cyklu se počet přistupujících uživatel zvýší o velikost zadaného kroku až po maximální počet uživatel zadaných testovací osobou.

Tento způsob testování ukazuje obrázek 8.5 a obrázek 8.6. Oba obrázky ukazují výsledek naprosto stejného testu, s tím rozdílem, že obrázek 4 zobrazuje výsledek testu po zapnutí programu, v tomto případě musí být před zavoláním metody z webové služby navázáno spojení mezi klientem a webovou službou/serverem a musí být zkompilována webová služba, toto se projeví na prvním cyklu testu (v tomto případě volání metody jedním uživatelem), ten proto trvá o mnoho déle, než v případě, že by byla metoda pouze volána. Pro srovnání byl tento test proveden ještě jedenkrát. Jak je vidět z rozdílu výsledků z obrázku 8.5 a 8.6, toto navázání spojení a zkreslení trvalo 287 ms.



Obrázek 8.5 Test provedený po spuštění aplikace



Obrázek 8.6 Test bez režie webové služby

První cyklus prvního měření byl tedy pro použití do statistik nebo průměrování hodnot měření nepoužitelný, na druhou stranu vidět dobu navázání spojení severu s webovou službou je zajímavé. Aby se nemohlo stát, že při testování opomeneme vynechat test zahrnující počáteční režii webové služby, která by výsledky testů nepříjemně ovlivnila, byla aplikace ještě trošku pozměněna, tak aby se veškerá režie webové služby neprojevila v testovacích datech, je od nich tedy odečtena a pro informaci připsána v textboxu.

V případě zvolení random způsobu generování pacientů, bude náhodně zvolen počet uživatelů v rozmezí minimálního až maximálního počtu uživatelů, zadaného testující osobou a tímto nasimulovaným počtem uživatelů bude přistoupeno ke zvolené metodě. Tento postup se bude opakovat tak dlouho, dokud nevyprší doba testování zadaná testující osobou. Jakmile uplyne doba testování, dokončí se právě prováděné volání metody, ale další kolo testu již spuštěno nebude.

Databáze byla pro ukázku uložena na dvou různých strojích a na obou byla měřena velikost odezvy. První, výkonnější PC má procesor 2x4GHz a paměť 6GB DDR2 RAM, druhý PC obsahuje procesor AMD 850 MHz a paměť 512MB SD RAM.

Typy provedených testů:

Způsob generování uživatelů	Rozsah uživatelů	Doba testování/Krok
2x4GHz, 6GB DDR2 RAM		
Postupně	1 – 30	Krok = 1
Postupně	1 – 1000	Krok = 50
Postupně	100 – 200	Krok = 10
Postupně	200 – 500	Krok = 20
Postupně	500 – 800	Krok = 20
Postupně	500 – 1000	Krok = 50
Postupně	800 – 1000	Krok = 10
Random	1 – 100	30 s, 60 s, 120 s
Random	1 – 200	30 s, 60 s, 120 s
Random	1 – 500	3 min, 5 min, 10 min
Random	1 – 700	3 min, 5 min, 10 min
Random	1 – 800	3 min, 5 min, 10 min

Random	1 – 900	3 min, 5 min, 10 min
Random	1 – 1000	3 min, 5 min, 10 min
Random	1000	10 min
AMD 850MHz, 512MB SD RAM		
Postupně	1 – 30	Krok = 1
Postupně	1 – 1000	Krok = 50
Postupně	100 – 200	Krok = 10
Postupně	200 – 500	Krok = 20
Postupně	500 – 800	Krok = 20
Postupně	500 – 1000	Krok = 50
Postupně	800 – 1000	Krok = 10
Random	1 – 200	120 s
Random	1 – 700	10 min
Random	1 – 1000	10 min
Random	1000	10 min

Tabulka 3 Typy provedených testů

Každý typ testů uvedených v tabulce výše je proveden v pěti iteracích, veškeré hodnoty jsou poté zpracovány do přehledných tabulek. Pro testy, při nichž jsou uživatelé generováni postupně, jsou z těchto hodnot následně počítány hodnoty „MODUS“, „MEDIAN“ a „Aritmetický průměr“. Hodnota MODUS udává nejčastěji se vyskytující hodnotu v provedených měřeních. Jelikož bylo měřeno pouze 5 iterací, ne vždy se vyskytly shodné hodnoty měření, proto některé položky vypočtených hodnot obsahují hodnotu #N/A. K provedení více iterací měření by však bylo zapotřebí mnohem více času na dokončení testů a jejich vyhodnocení. Naše měření však nejsou na přesnost a opakovatelnost hodnot kritická a odlišnost jednotlivých měření o několik milisekund pro nás není podstatné, jedná se pouze o informativní měření, zjišťující jakou dobu potřebuje server na odpověď na požadavek, která je navíc ovlivňována momentální vytížeností serveru.

Výsledky testů:

- 500 – 1000 uživatelů, generováni postupně
  Tabulka naměřených hodnot

		2x4GHz + 6GB DDR2 RAM (Server)					AMD 850MHz + 512MB SD RAM (PC)				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
500	get_User	892	888	895	1176	1078	3511	3570	3526	3526	3535
550	get_User	979	1006	994	1239	1239	3861	3898	3886	3852	3887
600	get_User	1071	1076	1083	1348	1243	4212	4255	4184	4270	4192
650	get_User	1168	1154	1182	1438	1348	4574	4543	4574	4588	4548
700	get_User	1254	1265	1240	1617	1443	4927	4967	4942	4933	4911
750	get_User	1342	1346	1354	1591	1549	5289	5309	5231	5281	5272
800	get_User	1441	1425	1662	1873	1682	5616	5645	5635	5657	5696
850	get_User	1595	1526	1964	1804	1656	6023	5939	5994	5974	5978
900	get_User	1613	1604	1986	1740	1854	6344	6316	6238	6387	6324
950	get_User	1703	1705	1874	1725	1914	6643	6677	6689	6707	6713
1000	get_User	1797	1786	2434	1796	2072	7037	7058	7050	7010	7079

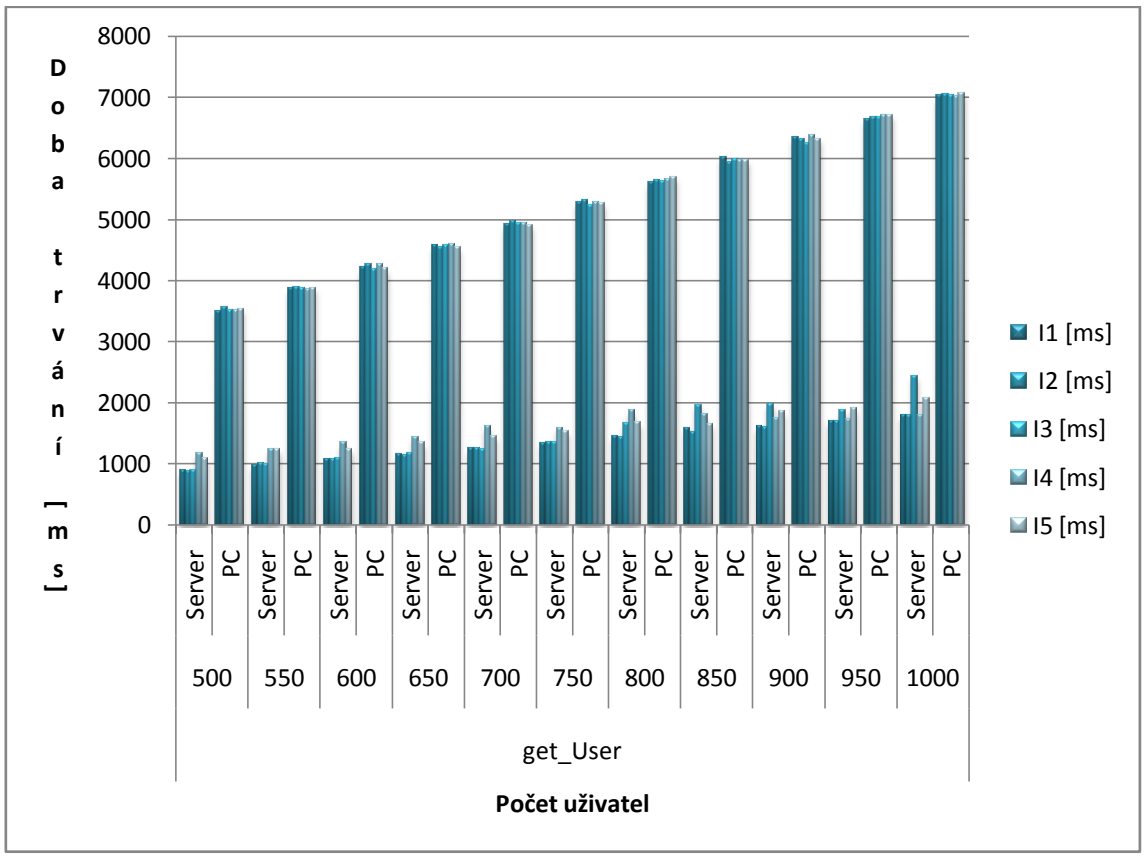
Tabulka 4 Naměřené hodnoty pro 500 - 1000 uživatel

Tabulka vypočtených hodnot

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
500	#N/A	895	985,8	3526	3526	3533,6
650	#N/A	1182	1258	4574	4574	4565,4
800	#N/A	1662	1616,6	#N/A	5645	5649,8
1000	#N/A	1797	1977	#N/A	7050	7046,8

Tabulka 5 Vypočtené hodnoty pro 500 - 1000 uživatel

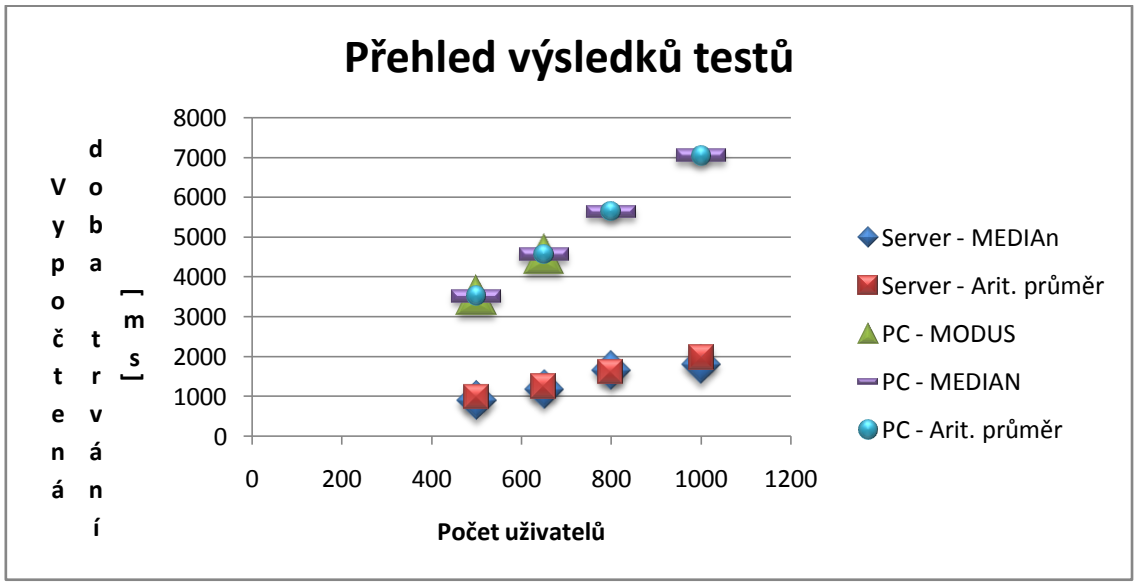
Graf naměřených hodnot se srovnáním obou použitých PC



Obrázek 8.7 Kontingenční graf pro 500 - 1000 uživatel

Graf ukazuje, jak se postupně zvyšuje odezva měření se zvyšujícím se počtem přistupujících uživatelů. Z jednotlivých sloupců je vidět, že hodnoty v každé iteraci pro stejný počet uživatelů mírně kolísají a to pro oba typy strojů, které jsou pro srovnání zobrazeny vždy pro daný počet uživatelů vedle sebe. Kolísání oněch hodnot může být dáno přepínáním mezi procesy na serveru, ten nemusí být schopen obsloužit náš požadavek vždy přesně ve chvíli, kdy jej obdrží, také to může být latencí sítě atp. Jsou to však opravdu malé odlišnosti. Rozdíl mezi jednotlivými stroji se zvyšuje se zvyšujícím se počtem simultánních přístupů. Při pohledu na graf je totiž vidět, že pro 500 uživatel je rozdíl mezi stroji přibližně 2.5 s, pro 1000 uživatel je však tento rozdíl i 5s.

Graf vypočtených hodnot



Obrázek 8.8 Graf vypočtených hodnot pro 500 - 1000 uživatelů

Vypočtené hodnoty i graf těchto hodnot prezentuje, jak se tyto hodnoty sw zvyšujícím se počtem uživatel zvyšují i tyto hodnot. Jednotlivé vypočtené hodnoty (Median, modus a průměr) se pro jeden typ stroje a stejný počet uživatel téměř překrývají, jsou opravdu téměř shodné, což můžete vidět i v tabulce těchto hodnot. Toto by se změnilo například, pokud by bylo pro výpočet použito více iterací, například pro aritmetický průmět by totiž stačila jedna velmi odlišná hodnota a tato hodnota by se mohla rapidně zvětšit. V tomto případě však byly hodnoty v jednotlivých iteracích velmi podobné a proto jsou i vypočtené hodnoty přibližně stejné.

- 1000 – 1000 uživatelů, generováni random

Tabulka naměřených hodnot

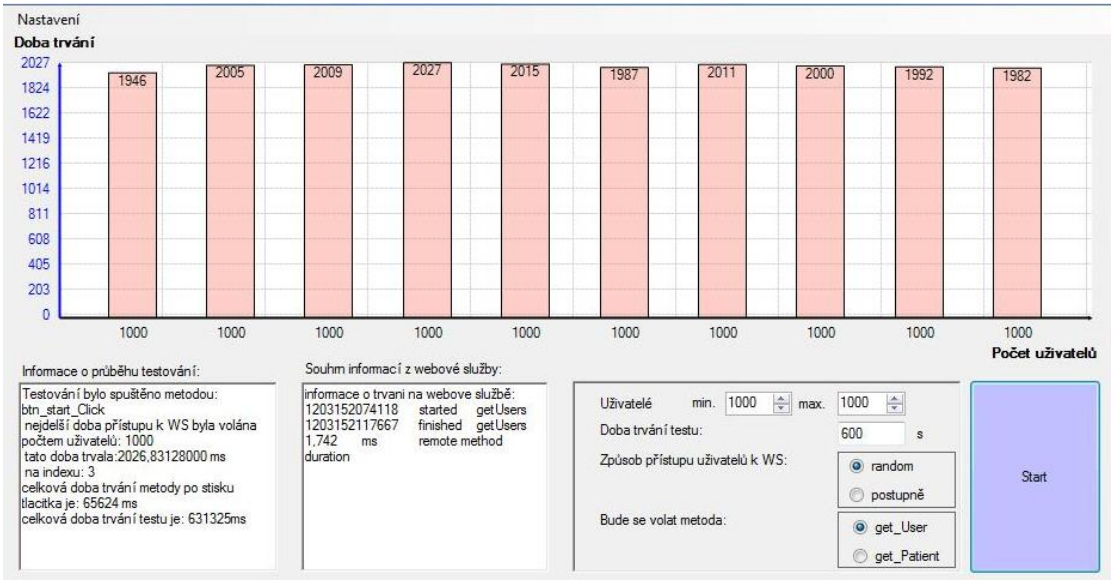
doba testovani: 10 min		2x4GHz + 6GB DDR2 RAM				
		IT1	IT2	IT3	IT4	IT5
volana metoda	poc. Uziv.	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]
get_User	1000	1901	1966	1960	1908	1946
get_User	1000	1924	1933	1933	1922	2005
get_User	1000	1901	1952	1952	1879	2009
get_User	1000	2048	1956	1956	1914	2027
get_User	1000	1949	1971	1971	1902	2015
get_User	1000	1914	1918	1918	1972	2987
get_User	1000	1876	1937	1937	1903	2011
get_User	1000	1950			1914	2000
get_User						1992
get_User						1982

Tabulka 6 Naměřené hodnoty pro 1000 uživatel

AMD 850MHz + 512MB SD RAM				
IT1	IT2	IT3	IT4	IT5
doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]
7069	7106	7078	7103	7021
6039	7068	7049	7001	6981
6976	7005	6976	6987	6979
6980	7061	7050	6952	7004
7000	7011	7024	7033	7060
7046	7030	7084	7130	6983
7004	7054	7020		7037
7009	7071			7019

Tabulka 7 Vypočtené hodnoty pro 1000 uživatel

Grafy naměřených hodnot



Obrázek 8.9 Graf výsledků pro stroj typu server - Iterace č. 5

Na obrázku 8.9 je výstupní graf z testovacího programu. Tento graf byl naměřen na databázi uložené na PC s procesorem 2x2 GHz s pamětí 6GB, ukazuje, jak se mění odezvy při dotazování tisícem uživatelů simultánně po dobu 10ti minut. Jak graf ukazuje hodnoty se opět liší jen minimálně, pohybují se stále přibližně kolem 2 sekund. Mírné odchylky mohou být způsobeny stejnými příčinami, jak toho bylo u postupného měření (obrázek 8.7). Pro toto měření bylo opět provedeno 5 iterací testů, obrázek 8.9 je pouze jednou z těchto iterací. Ostatní grafy najdete v příloze IV, této práce.

Obrázek 8.10 Výsledek měření pro stroj typu PC - Iterace č. 5

Obrázek 8.10 zobrauje stejné měření jako obrázek 8.9, toto měření však bylo provedeno na databázi uložené na PC s procesorem AMD 850 MHz a paměti 512 MB. Doba odezvy pro jednotlivá volání, se stejně jako na předchozím obrázku mírně liší. Jediným rozdílem v těchto měřeních je pouze to, že tento pomalejší stroj má odezvy dlouhé kolem sedmi sekund, což je oproti minulému měření o 5 sekund více. Ostatní iterace tohoto měření naleznete opět v příloze IV

Výsledky provedeného měření ukazují, že zatížení webové služby s růstem počtu přistupujících uživatelů stoupá v řádech jednotek milisekund. Z kontingenčního grafu (obrázek 8.6) je vidět, že při jednotlivých iteracích měření, hodnoty kolísají, ale drží se přibližně stejných hodnot pro stejné množství přistupujících uživatelů, liší se v řádech milisekund až desítek milisekund, což je pro množství přistupujících uživatelů přípustná změna. Dojde – li k větší odchylce, jako například na obrázku 8.6, kde délka odezvy pro 19 uživatelů je 34 ms a pro 20 uživatelů již 57 ms což je na toto množství simultánních přístupů velký rozdíl, to může být způsobeno tím, že serveru v danou chvíli přišel požadavek na vykonání jiného, důležitějšího procesu a proto nebyl náš požadavek vykonán okamžitě. Při pohledu na obrázek 8.5, je pro srovnání rozdíl v odezvách pro 19 a 20 uživatel pouze 2 ms.

Veškeré tabulky naměřených hodnot a z nich plynoucí grafy nelze na toto místo z důvodu jejich množství a rozsáhlosti umístit, zde je uvedena jen malá část z nich, zbytek je vložen do přílohy číslo IV tohoto dokumentu.

Testování formou takzvané „bílé skříňky“ tato práce neobsahuje, provádí si jej vývojář aplikace.

9 Diskuze výsledků

Účelem diplomové práce bylo zpracování systémových požadavků agentury domácí péče pro následný vývoj aplikace, která má být modernizací nynějšího systému postaveného na Access databázi. Požadavkem agentury bylo vytvoření offline systému, který umožňuje správu dat pacientů a zaměstnanců agentury, správu veškerých číselníků potřebných pro chod agentury a účtování s pojišťovnou a hlavně plánování úkonů prováděných pacientům agentury. Součástí systému je také mobilní aplikace, která zaměstnancům umožní upravovat záznamy pacientům přímo v terénu, zobrazovat si své úkoly a po jejich provedení je ukončit. Veškeré požadavky byly sepsány do podrobné systémové specifikace.

Pro úplnost je systémová specifikace doplněna o analytické diagramy. Jedná se o objektově orientovanou analýzu pomocí jazyka UML a jím specifikovanými typy diagramů. Pro získání dostatečného popisu vyvíjeného systému pomocí diagramů, bylo v této práci použito čtyř typů UML diagramů a to Use case diagram, který zobrazuje celkovou funkcionalitu systému, a to pomocí jednotlivých případů užití a jejich scénářů, aktivity diagram popisující jednotlivé případy užití z pohledu jednotlivých akcí potřebných ke splnění funkce, sekvenční diagram, který tyto případy užití popisuje s ohledem na objekty a zprávy, které si mezi sebou v časové ose předávají. Poslední použitý typ diagramů je diagram tříd, ten vyvíjený systém rozdělí do tříd a jejich vlastností a operací, které bude vývojář potřebovat k implementaci systému.

Systém byl vyvíjen dle spirálového modelu životního cyklu systému, tudíž při implementaci byla do systému v každé iteraci podle časového plánu nabalována další funkcionalita. Systém byl navržen tak, že je složen ze 4 komponent, z toho dvě komponenty jsou uživatelští klienti, PC klient běžící jako desktop aplikace pro správu dat a číselníků, mobilní klient běžící na PDA, Server uchovávající veškerá potřebná data pro práci a synchronizační rozhraní, které zajišťuje synchronizaci mobilního zařízení s daty uloženými na serveru. Veškeré komponenty pracují a jsou vyvíjeny na platformách firmy Microsoft a to Microsoft SQL Server a Microsoft Visual Studio 2008, proto také ke své práci veškeré komponenty potřebují .NET Framework/Compact Framework 3.5 a operační systém Windows XP nebo Vista.

Dále bylo v rámci diplomové práce potřeba navrhnout databázi, která poběží na serveru. Databáze je navržena tak, aby nemusela být žádná data obsažena fyzicky ve více tabulkách. Potřebuje – li více tabulek jedna data, jsou na ně odkázány unikátním primárním klíčem, tím bylo zabráněno tomu, aby při změně dat bylo potřeba jejich přepsání ve všech tabulkách, které s nimi pracují. Celkem tedy databáze obsahuje 20 tabulek, systém je však modulární, takže počet tabulek se může měnit podle potřeby použití či nepoužití některého z modulů. Přístup k datům, je realizován pomocí technologie LINQ.

Systém byl nakonec testován. Jako první bylo testováno, zda systém odpovídá systémové specifikaci. Po prozkoumání bylo zjištěno, že systém specifikaci odpovídá, bylo nalezeno jen pár nesrovnalostí nebo by se dalo říci doplnění specifikace, jako například, že položky číselníku

je z důvodu možných kolizí možné odstranit pouze v případě, že není nikde použita (s výjimkou historie), dále byly z důvodu časové náročnosti tohoto projektu a nutnosti dokončení prací z této verze systému v rozporu se specifikací odebrána funkce práce s dietami a skupinami, s těmito číselníky se však počítá v další verzi systému, jelikož práce na projektu bude pokračovat. Tyto změny jsme si mohli dovolit zahrnout, jelikož funkce, které nebyly implementovány, nepotřebuje agentura k použití hned. Tato změna tedy v žádném případě neovlivní nasazení systému do agentury k otestování již zaměstnanci, jejich připomínky budou následně zhodnoceny a bude – li to možné, implementovány do další verze systému. Funkce systému „Práce s testy a statistikami“, která není v tuto chvíli popsána ani v systémové specifikaci, bude implementována až po zadání požadávků na tuto funkci agenturou.

Systém byl vyvíjen agilně, tudíž upravení funkčnosti s ohledem na dodržení časového plánu je žádoucí, navíc funkce, které byly vynechány, nejsou pro práci agentury kritické. Při dalším testování byly vytvořeny scénáře jeho použití a zjišťováno, zda systém vykoná, co od něj uživatel očekává. Poté byly sepsány funkce, které mají být pro určité typy rolí zakázány, a bylo určováno, zda po přihlášení uživatele dané role, má tento uživatel přístup opravdu pouze k těm funkcím, které jsou mu povoleny. Tyto testy byly vyhodnoceny kladně, což znamená, že aplikace vytvořena diplomovou prací navazující na tuto, funguje správně v rozsahu prototypové aplikace. Do testování uživatelského ozhraní bylo zahrnuto také posouzení vzhledu a intuitivnosti ovládání aplikace. Ta byla shledána za příjemné prostředí pro práci a její ovládání bylo uznáno dostatečně intuitivním. Rozdělení do dvou hlavních objektů a jejich podobjektů je pro práci zdravotních sester, neznalých mnoha informačních systémů velice přehledné.

Při testování uživatelského rozhraní však bylo zjištěno, že v případě, že se uživatel nebude držet správného používání aplikace, namísto toho bude pracovat intuitivně, jako každý uživatel, může v nynější aplikaci dojít ke kolizím. Například při vytvoření plánu uživateli dojde, že daná položka v úkolu není potřebná, přepne se do tohoto tabu, odstraní položku, tato položka však může zůstat v nedokončeném plánu a při jeho uložení systém zkolabuje, jelikož položka, na kterou se daný plán s danou diagnózou odkazuje, již nebude dostupná. To by mohlo být při užívání systému velkým problémem a z tohoto důvodu bylo přistoupeno, i přes zásadní nedostatek času, k vytvoření nového uživatelského rozhraní a betaverze aplikace, ve kterém již nebyly veškeré editace otevírány v jednotlivých tabech, ale pro editaci je vždy otevřen modální dialog. Jiné zásadní změny v ovládání aplikace provedeny nebyly, ovládání nové verze aplikace je tedy stejně přívětivé a intuitivní, jako ovládání předchozí verze. Pro novou verzi systému prozatím neexistuje specifikace, jelikož upřesňujíci požadavky byly dodány v době uzavírání diplomové práce.

Nakonec byl systém testován ze zátěžového hlediska. K tomuto testu byla vytvořena aplikace, která umožňuje měření doby odezvy SQL serveru na dotaz v závislosti na počtu simultánních přístupů k datům. Aplikace byla vytvořena ve vývojovém prostředí Microsoft Visual Studio 2008, přistupuje k databázi pomocí webové služby. Tato možnost byla testována z důvodu

potřeby změřit délku odezev, jelikož má být webová služba do aplikace implementována. Aplikace umožňuje měřit dobu odezvy databázového serveru, ale také dobu, kterou trvá doprava požadavku k serveru, zabalení kolekce dat a jejich následná doprava k uživateli, zpracování dat v aplikaci k této odezvě není zahrnuto. Pro zajímavost aplikace uživatele informuje také o režii webové služby, kterou má na zpoždění vykonání uživatelského požadavku. Toto zpoždění zahrnuje navázání spojení s databází a kompilaci webové služby. Data získána tímto měřením jsou prezentována ve sloupcovém grafu, v tabulkách a shrnuta v textových polích. Z výsledků bylo patrné, že délka odezvy úměrně stoupá se zvyšujícím se počtem uživatelů a ani při 1000 uživatelů neměla databáze problém požadavek zpracovat, což pro účely agentury nadměrně dostačuje. Na délku odezvy má však velký vliv stroj, na kterém je databázový server uložen, výsledky testů ukazují, že při použití stroje s pamětí pouhých 512MB a procesorem 850MHz, dosahují odezvy při 1000 uživatelů odezev až 7s, což je v rozporu se specifikací, ta udává maximální odezvu 5s. Dále je potřeba počítat s tím, že odezva poroste také se zvyšujícím se počtem záznamů v databázi. Stroj, který je však použit jako databáze má procesor 2x2GHz a paměť 6GB, odezvy získané při testování na tomto stroji dosahovali maximálně 2s, což požadavkům vyhovuje a při tak velkém počtu simultánních přístupů by má rezervu, která by mohla pokrýt zvýšený počet záznamů. Délku odezvy však mimo jiné mírně ovlivňuje mnoho aspektů, server nemusí být zrovna ve chvíli, kdy příjde náš požadavek připraven požadavek spracovat, dále může být odezva ovlivněna latencí sítě. Tyto aspekty dobu odezvy ovlivní, ne však zásadně, jedná se pouze o mírné kolísání doby odezvy v řádech milisekund pro malý počet uživatelů a desítky milisekund pro větší počet uživatel.

Aplikace je modulární a postavena tak, aby při minimálních změnách byla schopna vyhovět požadavkům jiných agentur pro správu dat a to při minimálním zásahu do implementace systému.

Závěr

Cílem mé práce bylo vytvoření systémové specifikace systému vyvíjeného pro agenturu domácí péče, jeho návrh, analýza a testování. Systém měl být navržen k tomu, aby sloužil pro správu dat pacientů a zaměstnanců agentury, plánování úkonů pacientům a správu materiálů potřebnou k vykonání úkonu. Specifikace byla sepsána z požadavků agetury do přesné a přehledné podoby, určující výsledný vzhled a funkčnost systému.

Analýza systému byla provedena oběktově orientovanou analýzou pomocí jazyka UML (Unified Model Language). Pro potřeby systému jsem si při analýze systému vystačila se čtyřmi typy diagramů, 2 typy popisují celý systém, jsou to Use Case diagram a diagram tříd. Další dva typy diagramů jsou použity pro popis každé funkce systému, aby bylo vidět, jakým způsobem každá z funkcí pracuje. Diagramy pro popis funkčnosti systému jsou aktivity a sekvenční diagramy.

V návrhu systému bylo zapotřebí udělat několik věcí, a to navrhnout architekturu systému a databázi, která bude uchovávat data, se kterými potřebuje systém a agentura pracovat. Architektura má 4 části, PC klienta pro správu veškerých dat, Mobilního klienta pro zobrazování a ukončování úkolu zaměstnancem v terénu, synchronizační rozhraní pro synchronizaci PDA zařízení s PC klientem. Poslední část je databázový server, ten byl navržen tak, že každý objekt používaný následně při implementaci je v databázi složen z více tabulek. Tím bylo zamezeno tomu, aby byla pro každý objekt jen jedna tabulka, která by obsahovala duplicitní data.

Posledním řešeným problémem této práce bylo testování systému, které mělo ověřit správnost a bezchybnost vyvinutého systému. Výsledky testování systémová specifikace a scénářů použití systému, včetně testování funkčnosti přístupu rolí k různým datům vyšly kladně. Při zátěžovém testování systému bylo ověřeno, že stroj zvolený jako server, vyhovuje specifikačnímu požadavku s ohledem na požadovanou dobu odezvy, ta je vyhovující i při simultánním přístupu 1000 uživatelů. Doba odezvy byla v tomto případě 2s, maximální povolená odezva je 5s. Dále bylo tímto způsobem ověřeno, že systém nebude mít problém v případě simultánních přístupů více uživatelů.

Systém je ve vyhovujícím a funkčním stavu a může být zaveden k testování uživateli v agentuře.

Extended abstrakt

This project deal with system development for data management and scheduling of personal sources in home care agencies. Development is buckthorn from analyses, over proposal, implementation and resulting testing application, thereby try cast over whole problems software engineering. System consists of the three main parts – PC application, mobile application and database server. All these parts are built on technology Microsoft .NET solution namely on performances .NET Framework 3.5 and MS SQL server 2008 express edition for PC application, .NET Compact Framework 3.5 and MS SQL server 3.5 Compact Edition whereon be created database server for mobile application. System works offline, whereas synchronization mobile arrangement with data in database is made only in agency by the help of technology USB, Bluetooth or WiFi. Software layer synchronization administration data is based on technologies ADO .NET and Linq. System will further develop and will base for advancement biomedical science research framework.

Introduction:

The project is made on requirements of HomeCare Agency Ostrava, Czech republic for use with mobile medical health personal.

At present time the system specifications and first round of implementation is ready. The developed system is made on the basis needs gained from submitter (HomeCare Agency Ostrava, Czech Republic), as well as on the basis analyses current solving plant MS ACCESS 98. Globally the pilot desktop client and mobile client is implemented. The data and the database model appears to be final, whereas to definite modifications in face of original specifications. Data strobe be functional, but requires a lot of adjustments for finalization to the final state.

Analysis and proposal of the system:

First step before development is a specification with follow-up analysis and proposal of the solution. These paces are practiced by the consultation with customer/or by different method of collection of demands and are deal with detail text form (systemic specification) with supplemented UML diagrams. We get along with recommended processes for analysis and proposal. The system is described by static and dynamic diagrams.

General analysis and proposal:

Contextual diagram – view on the system as unit, and its communication with external systems. In this case an external system realizes roles of users using the system. In the form how are the users displayed on the diagram the system will be delivered. There´s is possibility to adapt the rights to the roles according to needs, so maybe that during the using of the system the diagram already won´t be valid.

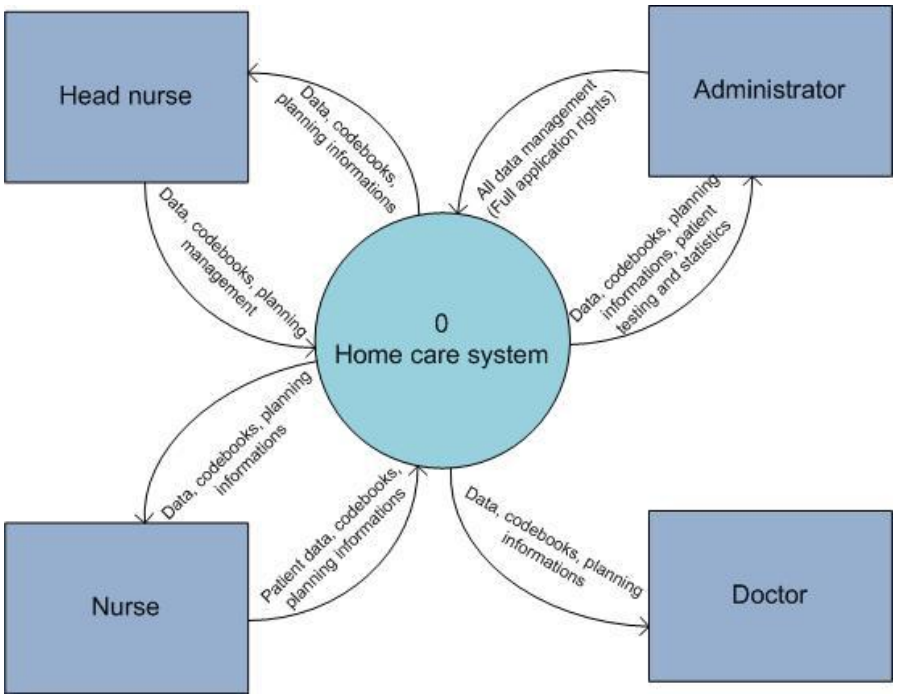


Figure 1 Context diagram of developed system.

The system will contain 4 types of users:

- Administrator – the main user of the system and the one whose rights can´t be modified. He has all rights in the system.

- Head sister – the rights of this role can be freely modified by the administrator. Basic rights of the head sister are to read from the system and to register in it the information's about all dials, data and work plans.

- Nurse – she has right to read from the system the same information as the head sister, but she can write to the system only data of the patients.

- Doctor – towards to the system has rights only to count, he can´t influence the system in neither way as well as he isn´t allowed to write anything.

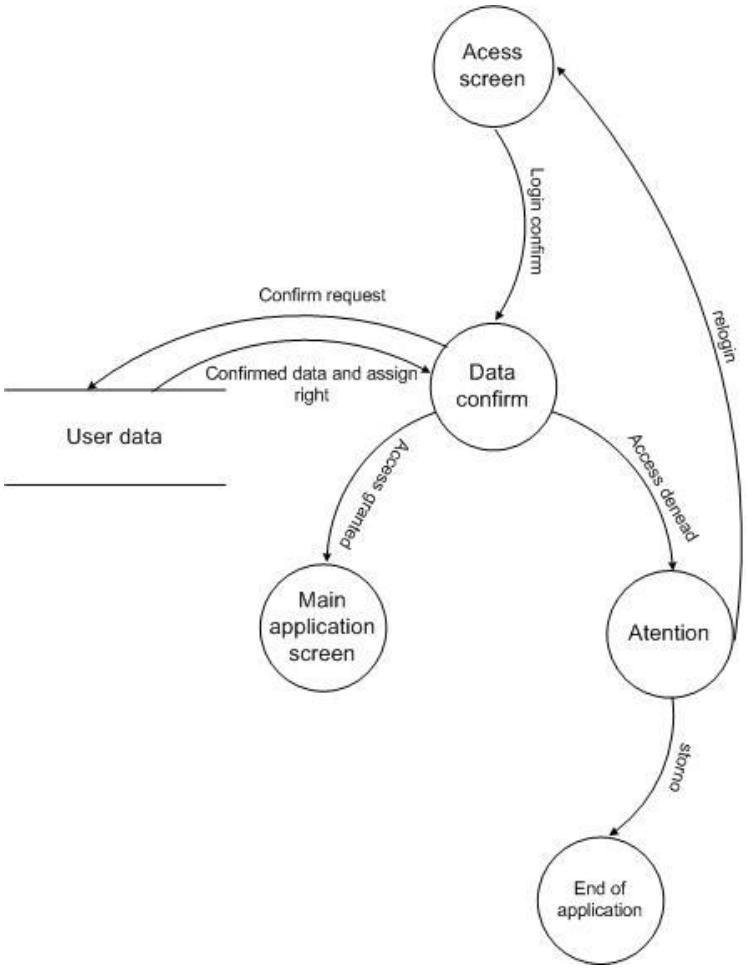


Figure 2 Data flow diagram of developed system.

Data flow diagram [Fig. 2] describes the flow of data from process to process at the sign on of the user, obtaining of the information about the role which is assigned him and of the follow-up scheme of the main windows of the application. After the loading of the information of entered user there in the main windows will open up only the functions which has the user right to manage. If the user will input enter data which differ from the data in database, the database will send out information about non-finding of the data. Then the user can enter again or to finish the application.

Showed sequential diagram [Fig. 3] describes the effect of login of the user from the view of the objects and messages which they exchange between them.

The communication between the single objects pass simultaneously, it means that every objects will send a message to other object and wait for an answer from it. In the time of waiting it doesn´t practice any operations.

The data proposal:

The data model considerably differs from the data model of the objects used in the application and also from the model proposed in MS Access. The original model contained 3 tables whose undesirable character was the duplication of the useless data. The new model try to work with data store more effectively, therefore the data are divided into more tables so that the notation which is used several times in the same form would be saved just once.

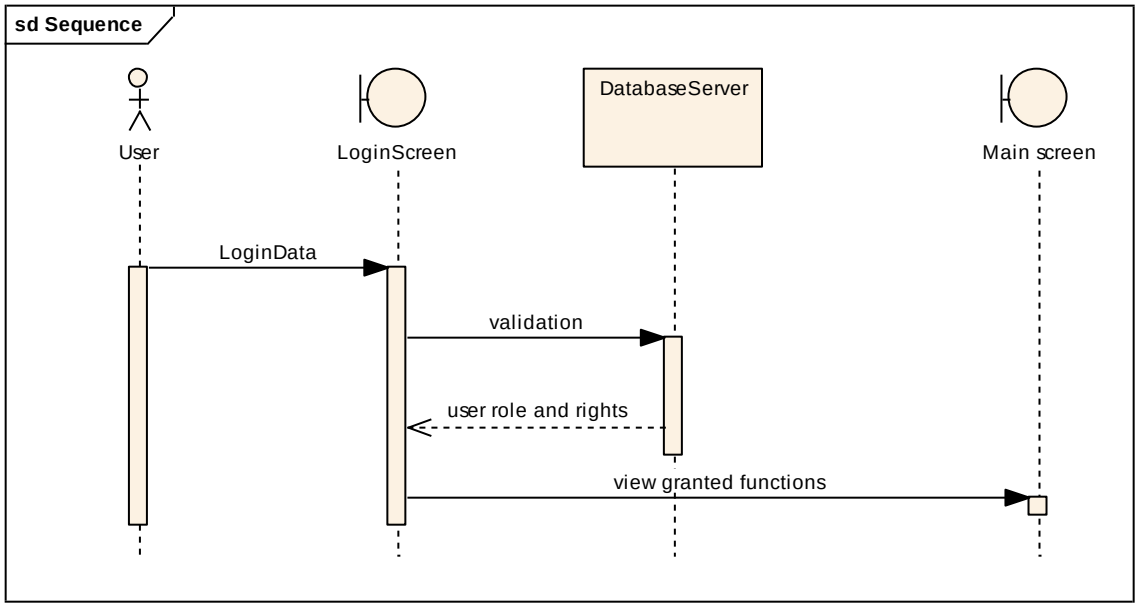


Figure 3 Sequential diagram of developed system.

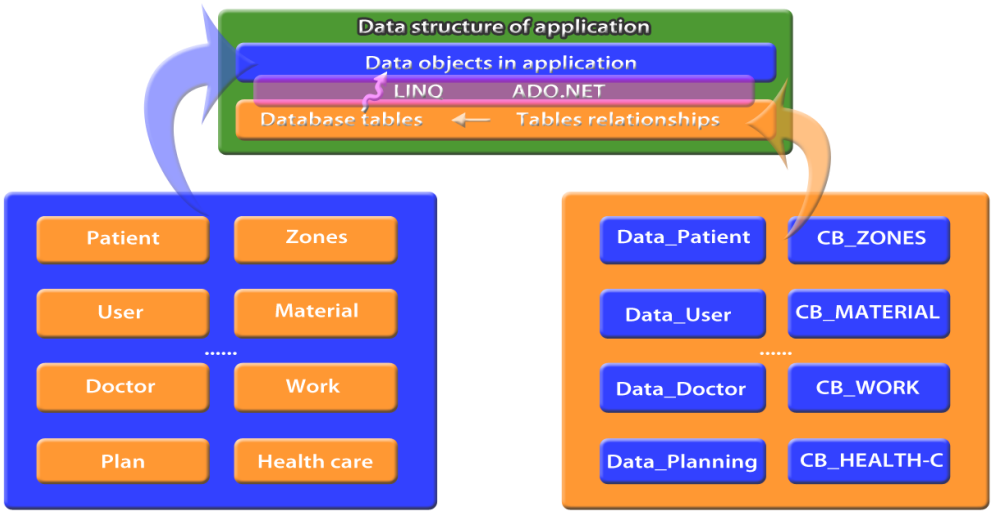


Figure 4 Data architecture from database to GUI data reprentation.

The model was also extended about another entities – implementation of the dials of materials, zones, duties etc. the particular tables are tied to the groups of relations..

The architecture of the system:

The resulting system composes from 3 basic parts which are interconnected. Each from the parts could be involve to other block of the suggested model for proposal of the architecture – models MVC (Model View Controller). The possibilities of the client application are far wider than the possibilities of PDA application. This results from the limitation of architecture of the particular platforms and also of the demand on the system.

On the lowest ply there is a possibility to divide the system to the hardware and the software part. The hardware creates the server, particular client stations, mobile equipment and communication interface. The communication can pass by the aid of wire affiliation (intranet, USB cable) or wireless connection (Wireless LAN, Bluetooth). One of the demands was the operation of offline system – mobile clients can/will synchronize only in the network of PAN type which will be strictly separate of any local network or of the network internet. This solution reduces the possibilities of stealing of the data just on purely physical way. The server and client can be for the personal computer the same equipment. There can be used any mobile equipment which complete the demands for the operation of the system.

The software ply can be divided by 2 forms – division according to parts of the hardware, when will be practiced (the parts create the particular compacted blocks of the application) or according to aforesaid suggested model MVC.

Division according to the hardware/logical units:

- The server application – services – supplying of the data from the database
- The client application PC – standard application – work with the provided data and create interface machine – human.
- The client application PDA – standard mobile application – work with the available data in the terrain.

The division according the model MVC:

Model is the data ply built on the technology of keeping the data SQL and in the described system is represented by the technologies Linq, ADO.NET. These technologies create the data interface between the database and the data model of the very application – The data model isn´t accordant with the database model.

View represents the singles GUI applications and so the client applications for the personal PC and the mobile applications for the PDA. They graphically interprets the data with that is operating – they cover up the whole mechanism of the data processing.

Controller is the ply of the methods working on the interaction of the user interface according to just enabled actuating elements, resulting from the roles/ assigned rights and available data.

GUI of the platform:

Both GUI applications are proposed with respect on the objective group of the users, and therefore they try to minimize the number of paces to construction of the demanded work.

The client application for the PC uses the system of the markers which has hierarchic arrangement. On the highest ply there are 2 markers dividing application on the work with the data of patients and users and on the work with the dials. Particular markers are in the other level divided into the particular blocks of the entities. The last ply is the button evoking the particular operations over the applicable entity. The application serves to the administration of the users, patients, dials and the planning of the duties.

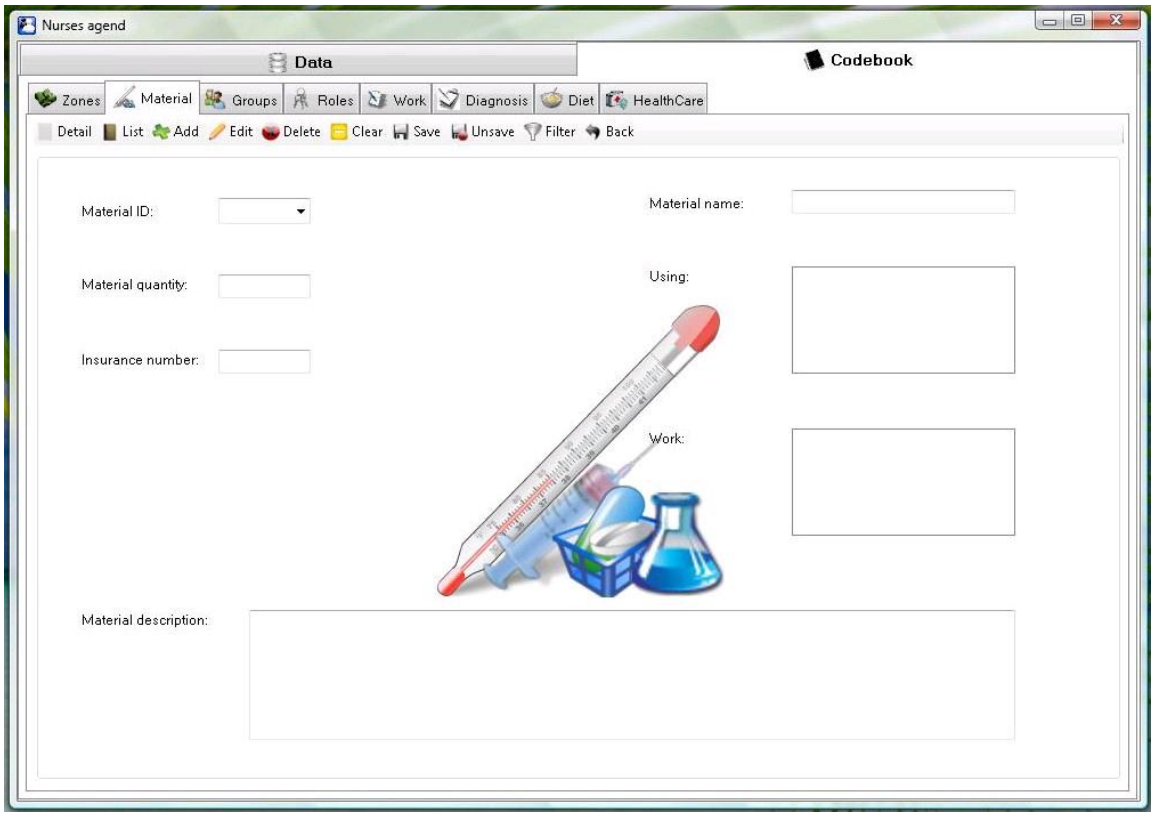


Figure 5 Desktop klient – Codebook managing

The mobile client is realized in different way where the control results from the different main sight of the application. Application serves to display of the duties and to enrollment of the comments. The control is arranged so that it would serve to the control of the classic application Windows Mobile.

Used technologies:

In whole solution there are used the technologies of the Microsoft company which should safe the trouble-free cooperation of the particular parts. The application is written in C# 3.0. language.

The basic architecture builds on use of the standard libraries available in NET Framework 3.5 for the desktop client part NET Compact Framework 3.5 for the mobile client application.

The graphical custom interface is built in both examples on the WinForms and the first available in the standard library.

The database machine practiced on the server is the MS SQL Server 2008 Express Edition which can be replaced by any different version of the MS SQL Server in version of 2008.

The ADO.NET technology is used for the synchronization of the data between the mobile and the server database. There in this part is also supposed the setting of the technology Linq. Relative new technology Linq supported in C# 3.0 is used like the interface in the both applications. The data from the generated classes are counting to the internal objects of the application for the easier work. Linq allows the single access to the same data from the different data sources by the same form, whereby they reduce the time of the progress and also the costs.

The datas saved on the memory card of the mobile equipment are coded by the internal mechanism OS Windows Mobile 6. It´s one of the easy form of the securing of the data.

Future work:

The particular parts will be needed to generality so that the system would be extendable. In other phases there will ne completed the functionality for the testing of the patients, generation of the statistics for the Statistic office and in not least series the extension of the displaying of the ache position of the employee in the terrain, with the aid of the map base of the company Seznam and the Google. One of the following pace after the fine-tuning if the application will be the creation of the Framework for the tele-medical applications. This framework should in them implement the general and the specific mechanisms, more bio-medical projects and in the future should facilitate the progress of the type similar bio-medical applications.

Figure 6 Mobile client screens.

Conclusions:

The present results are the conclusion of the diploma works that are proposing and implements the factual solution. The system could be marked as the prototypical solution.

Literatura

[1] http://www.fi.muni.cz/~smid/mis-zivcyk.htm (5.10. 2008)

[2] ČERNOHORSKÝ, Jindřich, Řídicí systémy s počítači, Skriptum VŠB – TUO.

[3] PATTON, Ron, Testování softwaru,Computer press, 2002

[4] WIEGERS, Karl, Požadavky na software, Computer press, 2008.

[5] http://mpavus.wz.cz/uml/uml-uvod-1.php (20.10. 2008)

[6] http://fpf.slu.cz/~vav10ui/obsahy/dipl/uml/Diagramy1.htm (20.10. 2008)

[7] ARLOW, Jim, NEUSTADT, Ila, *UML 2 a unifikovaný proces vývoje aplikací*, Computer press, 2008.

[8] JACOBSON, Ivar, NG, Pan-Wei, Aspect-Oriented Software Development with Use Cases, Addison Wesley Professional, 2004.

[9] Dokumentace k SW Enterprise Architect.

[10] http://www.osu.cz/katedry/kip/aktuality/sbornik99/mohlanec1.html (5.1. 2009)

[11] RUMBAUGH, James, JACOBSON, Ivar, BOOCH, Grady, The unified modeling language reference manual, Addison Wesley, 1999.

[12] http://en.wikipedia.org/wiki/Structured_analysis (13.2. 2009)

[13] http://cs.wikipedia.org/wiki/Diagram_datov%C3%BDch_tok%C5%AF (15.2. 2009)

[14] PECINOVKSÝ, Rudolf, Návrhové vzory, Computer press, 2007

[15] WIGLEY, Andy, WHEELWRIGHT, Stephen, Microsoft .NET Compact Framework, Microsoft press, 2003

[16] LACKO, Luboslav, Programujeme mobiln9 aplikace ve Visual Studiu .NET, Computer press, 2004

Příloha I Systémová specifikace

1 Úvod

1.1 Předmět specifikace

Tato specifikace popisuje požadavky na systém navrhovaný pro pečovatelskou službu, poskytující své služby jak přímo v sídle firmy, tak v domácnosti klientů. Účelem tohoto systému je převod stávajícího systému záznamů pacientů z klasické formy do elektronické podoby. Systém bude kromě elektronického uchovávání záznamů pacientů umožňovat také plánování úkolů zaměstnancům zaměstnavatelem pro daný den, přičemž si své úkoly zaměstnanec zobrazí ve svém pracovním mobilním zařízení.

1.2 Typografické konvence

Všechny prvky aplikace budou pojmenovány následovně:
zkratka názvu prvku_Název okna ve kterém se prvek nachází+Název účelu prvku

název typu prvku vždy začíná malým písmem a za ním píšeme vždy podtržítko, poté následuje zbytek názvu prvku, kde každé slovo začíná velkým písmem (Název prvku je psán strukturou psaní názvů, které se říká Camel).
například: máme – li v okně *Pacient* prvek typu *label* a je to prvek popisující Jméno pacienta, bude se tento prvek nazývat lbl_PatientsPatientName.

Všechny funkční požadavky mají stejnou prioritu.

1.3 Cílové publikum, návod ke čtení

Tento dokument je určen ke čtení analytikům, vývojářům aplikace, testerům a zadavateli (zákazníkovi).
Doporučené čtení specifikace je následující:

- Analytik a vývojář – čtou celou specifikaci.
- Tester – čte specifikaci podle typu testů.
- Zadavatel (zákazník) – čte obecný popis specifikace (část 2) a funkce systému (část 3).

1.4 Rozsah projektu

Účelem tohoto projektu je modernizace vedení záznamů pečovatelské služby z nynějšího papírového systému do přehlednější elektronické podoby. Data mají být uchovávána v databázi (database machine). Aplikace bude umožňovat správu zaměstnanců agentury, zákazníků - pacientů, plánování pracovních úkolů jednotlivým zaměstnancům prostřednictvím desktopového klienta, přičemž zaměstnanci si mohou zobrazit informace o svých denních úkolech na svých pracovních mobilních zařízeních (PDA). Aplikace bude mít implementovánu možnost generování statistik a testů pacientů z uchovaných dat. Dále aplikace umožňuje správu spotřebovaného materiálu. Přístup

k jednotlivým funkcím aplikace mají zaměstnanci povolen přidělením rolí, přičemž každá role bude mít jiná přístupová práva. Podrobněji se funkcemi aplikace zabýváme v kapitole “3“ této specifikace.

Vztah systému k uživatelům:

- **Vedoucí** x **Systém** - vedoucí pracovník má vůči systému za úkol (pravomoce) kontrolu provedených prací, generování statistik, ovlivňování požadavků a spravování číselníků.
- **Hlavní sestra** x **Systém** - přidávání, editace odstranění uživatele nebo pacienta, rozdělování úkolů zaměstnancům a kontrola jejich splnění, správa číselníků.
- **Zdravotní sestra** x **Systém** - zobrazení vlastních přiřazených úkolů, přidání editace a odstranění pacienta ze systému, zaznamenání již provedeného úkolu.
- **Pacient** x **Systém** - pacient systém ovlivňuje pouze pasivně.

Vztah systému k firemním cílům:

Agentura chce touto aplikací dospět k modernizaci záznamového systému. Přechod na elektronickou formu záznamů a správy úkolů jim umožní větší skladnost záznamů (ubude nutnost archivovat velké množství papírových záznamů o zákaznících „pacientech“ a také záznamů o spotřebovaném materiálu atd.). Agentura má za cíl také jednodušší a rychlejší orientaci nebo vyhledávání v záznamech, dále také lepší orientaci uživatelů v zadaných úkolech a kontrolu jejich provedení.

1.5 Odkazy

Dokumentace databáze, systémová specifikace mobilního klienta, dokumentace databáze mobilního klienta.

2 Obecný popis

2.1 Kontext systému

Jedná se o novou aplikaci, bez jakýchkoliv předchozích verzí. Je to aplikace skládající se ze čtyř komponent (celků) a to z „Serverová část systému“, „PC klient pro správu uživatelů a plánování“, „Synchronizační klient“ a z části „Mobilní klient“. Produkt je desktopová a mobilní aplikace s databázovým serverem uchovávajícím data.

2.2 Funkce systému

Funkce s daty

- Práce s uživatelem
- Práce s doktorem
- Práce s pacientem
- Práce s plány
- Práce s historií
- Práce s testy a statistikami

Funkce s číselníky

- Práce se zónami
- Práce s materiálem
- Práce se skupinami
- Práce s rolemi
- Práce s úkoly
- Práce s diagnózami
- Práce s dietami
- Práce se zdravotní péčí

Jednotlivé funkce jsou podrobněji rozepsány v kapitole 3 systémové specifikace.

2.3 Třídy uživatelů

Vedoucí systému (administrátor) – jedná se o vedoucího uživatele aplikace, mající veškerá hlavní práva potřebná ke správě agentury. To znamená kontrolu pracovníků agentury, správu všech uživatelů, doktorů i zákazníků agentury. Vedoucí může kromě uživatelů spravovat také veškeré číselníky potřebné k managmentu agentury a to převážně pro proplacení zdravotních zákroků pojišťovnou a k jednoduché dokumentaci spotřebovaného materiálu. Tento uživatel je oprávněn také k práci se statistikami a testy. Uživatel tohoto typu bude v systému jen jeden a nebude možné nijak měnit jeho nastavení jakýmkoliv uživatelem, včetně sama sebe.

Hl. sestra – tento uživatel je uživatel nadřízený všem zdravotním pracovníkům v agentuře. Uživatel je oprávněn provádět správu uživatelů (vytvoření, editaci, odstranění) a zákazníků agentury. Dále je oprávněn přidělovat úkoly zaměstnancům (plánovat procedury zákazníkům) a kontrolu jejich provedení. Uživatel hlavní sestra má také oprávnění spravovat číselníky, jejichž funkce byla popsána již u vedoucího systému.

Zdravotní sestra – tento uživatel neplní převážně žádné administrátorské funkce. Má právo správy číselníků, plánů, dat pacientů a Mobilního klienta (viz. Kapitola 2.4), pomocí obou klientů spravuje vlastní úkoly (zobrazuje jejich znění a po splnění je ukončuje). Dále může v případě potřeby předat některý ze svých úkolů jinému uživateli (např. v případě, že by tento úkol z vážných důvodů nebyl schopen splnit). Dále je uživatel v těchto dvou klientech schopen přidat nového pacienta nebo stávajícího pacienta editovat či odstranit.

Tyto třídy uživatelů jsou základní typy uživatelů dodané s aplikací. Zákazník není schopen vytvářet nové uživatele, ale vedoucímu systému je umožněno měnit jednotlivým uživatelům role, které má přiřazeny.

2.4 Provozní prostředí

Serverová část systému – Tato část poběží v kancelářském prostředí klienta používajícího produkt. K realizaci bude použito klasické stolní PC vybavené následujícími softwarovými prostředky – database machine(databázové prostředí – např. MS SQL Server express edition), .NET compact

framework 3.5, IIS (internetový informační server), OS (operační systém Windows XP/Vista, windows server).

PC klient pro správu uživatelů a plánování – Tato část, sloužící pro administrátora aplikace a pro ostatní zaměstnance, bude provozována stejně jako serverová část v kancelářském prostředí klienta. Klient bude spuštěn na stolním PC či notebooku. Pro správný běh klienta bude mít PC (notebook) nainstalován operační systém Windows XP/Vista a technologii .NET Framework 3.5.

Mobilní klient – Jedná se o přenosné zařízení používané zaměstnanci v terénu při vykonávání práce. Hardwarová část mobilního klienta se bude skládat z mobilních zařízení s operačním systémem (Smart phone) nebo z PDA zařízení obsahujících WiFi, mohou obsahovat také bluetooth, ten však není povinný. Povinné softwarové vybavení použitých mobilních zařízení je mobilní verze databázového prostředí (např. MS SQL Server Compact Edition 3.5), .NET Compact Framework 3.5 a operační systém Windows Mobile verze 5 a vyšší.

Synchronizační rozhraní – jedná se o uzavřenou lokální síť sloužící k synchronizaci dat mezi jednotlivými částmi systému. Komunikační rozhraní obsahuje WiFi access point.

2.5 Omezení návrhu a implementace

Aplikace bude napsána s použitím technologie .NET verze 3.5, programovací jazyk bude použit libovolný kompatibilní s .NETem. Databázová vrstva musí být vytvořena v prostředí MS SQL server, přičemž k neobjektovým datům bude přistupováno pomocí technologie LINQ. Uživatelské rozhraní bude vytvořeno buď pomocí frameworků Visual Studia nebo jako WPF. Veškerá zařízení ne kterých poběží aplikace musí obsahovat .NET framework 3.5 (mobilní zařízení .NET Compact framework 3.5). Mobilní zařízení musí také obsahovat Wi-Fi.

2.6 Uživatelská dokumentace

Veškerá dokumentace bude dodána ve formátu PDF, psána bude dle vytvořené šablony (viz. příloha).
Součástí dokumentace bude:

- Uživatelský manuál pro PC klienta
- Uživatelský manuál pro mobilního klienta
- Nápověda v aplikaci PC klienta
- Nápověda v aplikaci mobilního klienta

Dokumentace nebude obsahovat online nápovědy ani průvodce aplikací.

2.7 Předpoklady a závislosti

Každá funkce aplikace bude zpřístupněna pouze uživateli, který má možnost tyto funkce vykonávat (uživatel musí být přiřazen k roli, které je funkce, kterou chce provádět přiřazena - povolena). Funkce, které nemá daný uživatel povoleny, mu budou znepřístupněny ihned po přihlášení uživatele do systému.
Vývoj systému není závislý na žádných neovlivnitelných vnějších okolnostech.

3 Funkce systému

3.1 Práce s uživatelem

3.1.1 Popis a priorita

Tato funkce umožňuje spravovat data uživatelů (zaměstnanců agentury) a to převážně uchovávat jejich osobní a pracovní údaje, jejich zapsání, editaci a vymazání.

3.1.2 Události a odpovědi

- **Detail** – tato událost zobrazí podrobné informace o aktuálně vybraném uživateli. V tomto okně nebude možné žádné údaje přepsat. Není – li vybrán žádný uživatel, zobrazí aplikace dialogové okno s informací o nevybrání uživatele.
- **List** – zobrazí seznam všech uživatelů uložených v aplikaci.
- **Add** – po kliknutí na tuto událost se v aplikaci zobrazí nevyplněné okno detailu uživatele (uživatel vyplní údaje nového uživatele).
- **Edit** – Také zobrazí detail označeného uživatele, zde ale bude možné jakýkoliv údaj opravit.
- **Delete** – odstraní vybraného uživatele ze seznamů uživatelů (odstranění bude provedeno pouze příznakem, veškerá data fyzicky v databázi zůstanou, budou neviditelná pouze pro aplikaci).
- **Clear** – všechny údaje v položkách detailu budou vyčištěny (pouze v editaci nebo při tvoření uživatele, u obyčejného zobrazení detailu není možné data editovat, tudíž ani smazat).
- **Save** – uloží změny provedené v detailu uživatele (platí pro vytvoření nového uživatele, editaci a odstranění uživatele).
- **Unsave** – tato událost se vrátí k poslední uložené verzi okna (byly – li provedeny nějaké změny, nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.1.3 Funkční požadavky

3.1.3.1 Přidání uživatele (Add user)

Pro přidání nového uživatele do systému se zobrazí okno detailu uživatele (okno musí obsahovat všechny položky obsažené v databázových tabulkách uživatele viz. dokumentace databáze). Toto okno bude mít všechna pole s údaji uživatele nevyplněna. Pole údajů musí mít povolen zápis. Po vyplnění všech povinných údajů nového uživatele (povinné údaje nemají v databázi povolenu položku „not null“) je potřeba pro dokončení vytvoření nového uživatele údaje uložit.
Nejsou – li vyplněny všechny povinné údaje, zobrazí se po kliknutí na událost uložit dialogové okno s upozorněním, že je třeba vyplnit všechny povinné údaje. Povinné údaje musí být v okně přidání uživatele viditelně označeny. Uživateli nesmí být povoleno zadání číslic do položky „Jméno“ a „Příjmení“ a zadání písmen do položky „Telefonní číslo“ a „Zone ID“. V položce „Zone ID“ také nesmí být povoleno zadání vyššího čísla zóny, než je maximální pořadové číslo existujících zón v systému.

Je – li číslo zóny zadáno vyšší než povolené, zobrazí se uživateli dialogové okno s upozorněním, že je zadáno příliš velké číslo (toto dialogové okno bude obsahovat i nejvyšší možné použitelné číslo zóny). Po zadání nepovoleného znaku do pole „Jméno“, „Příjmení“ nebo „Telefonní číslo“ se uživateli také zobrazí upozornění o použití nepovoleného znaku.

3.1.3.2 Editace uživatele (Edit user)

K editaci uživatele bude možno se dostat dvěma způsoby a to buď ze seznamu všech uživatelů (Je – li jeden z uživatelů označen a poté uživatel klikne na událost „Edit“. Není – li žádný uživatel vybrán, zobrazí se po kliknutí na „Edit“ dialog s upozorněním,že je nejprve třeba vybrat uživatele, který má být editován.) nebo z okna detailu uživatele (opět kliknutím na „Edit“). Přistupujeme – li na editaci ze seznamu uživatelů, musí se zobrazit okno detailu uživatele, ovšem všechna pole údajů musí mít povolen zápis, aby bylo možné libovolný údaj upravit. Přistupujeme – li k editaci přímo z okna editace, musí se po kliknutí na „Edit“ změnit u všech polí s údaji atribut povolení zápisu z „false“ na „true“. I po editaci musí být u všech povinných položek údaj vyplněn. Do polí mohou být zapsány stejné znaky jako při vytvoření nového uživatele. Při špatném vyplnění údaje (nepovolený znak nebo hodnota údaje) budou zobrazeny stejná dialogová okna a stejná upozornění jako v předchozím případě (viz 3.1.3.1 Přidání uživatele).

3.1.3.3 Odstranění uživatele (Delete user)

Libovolný uživatel může být odstraněn výběrem uživatele v seznamu všech uživatelů a kliknutím na událost „Delete“ nebo kliknutím na tuto událost v detailu daného uživatele. Po kliknutí na „Delete“ se zobrazí dialogové okno s upozorněním, že bude daný uživatel smazán a zároveň bude dialog vyžadovat potvrzení smazání uživatele (kliknutím na tlačítko „Ano“ v dialogovém okně). Akci bude možno stornovat kliknutím na tlačítko „Storno“ (uživatel se vrátí zpět do okna ze kterého vycházel při kliknutí na „Delete“). Bude – li na událost kliknuto v seznamu uživatelů bez toho, aby byl předem vybrán uživatel,který má být smazán, zobrazí se dialogové okno s upozorněním, že je nejdříve nutno vybrat uživatele, který má být smazán.

3.1.3.4 Login

Okno přihlášení uživatele se zobrazí vždy o spuštění aplikace. Uživatel se přihlásí pomocí uživatelského jména a hesla, tyto údaje budou porovnány s údaji v databázi. Souhlasí – li údaje, zobrazí se okno aplikace se seznamem pacientů. Pokud údaje nesouhlasí, bude uživatel upozorněn, že byly zadány neplatné údaje a otázán, zda se chce pokusit přihlásit znovu. Chce – li se uživatel přihlásit znovu, zobrazí se opět okno pro přihlášení, jinak bude aplikace zavřena.

3.1.3.5 Logout

Odhlášení může uživatel provést z libovolného okna aplikace, tlačítko pro odhlášení bude zobrazeno v každém okně aplikace. Po odhlášení bude okno aplikace zavřeno a zobrazí se okno pro přihlášení uživatele.

3.2 Práce s doktorem

3.2.1 Popis a priorita

Funkce umožňuje spravovat data doktorů pracujících pro agenturu, jeho základní údaje, zdravotní specializaci a telefonní číslo, na které je možno volat v případě nutnosti.

3.2.2 Události a odpovědi

Události a odpovědi na ně, jsou v případě práce s daty doktorů shodné s událostmi při práci s uživateli uvedenými výše (viz. kapitola 3.1.2).

3.2.3 Funkční požadavky

3.2.3.1 Přidání doktora (add doctor)

Viz. kapitola 3.1.3.1

3.2.3.2 Editace doktora (edit doctor)

Viz. kapitola 3.1.3.2

3.2.3.3 Odstranění doktora (delete doctor)

Viz. kapitola 3.1.3.3

3.3 Práce s pacientem

3.3.1 Popis a priorita

Umožňuje správu osobních i zdravotních údajů pacienta, potřebných zajištění kvalitní zdravotní péče.

3.3.2 Události a odpovědi

Události pro správu pacienta jsou shodné s událostmi, které jsou použity pro správu uživatele i doktora (viz. kapitola 3.1.2).

3.3.3 Funkční požadavky

3.3.3.1 Přidání pacienta (add patient)

Vytvoření nového pacienta probíhá shodně s vytvořením nového uživatele či doktora (viz kapitola 3.1.3.1). Opět musí být ošetřeny veškeré vstupy proti zápisu neplatných znaků (písmena v položkách „telefonní číslo“, „Poštovní směrovací číslo“ a „číslo zdravotní pojišťovny“, číslice v položkách „jméno“, „příjmení“, „Stát“, „kontakt“). Také musí být zajištěno, aby do položek „ID zóny“,

„Diagnóza“, „Doktor“, „Typ zdravotní péče“, „Indikace“ a „Dieta“ bylo možné zadat pouze platné hodnoty položek, které jsou obsaženy v databázi. Při pokusu o zadání neplatné hodnoty musí být vždy zobrazeno upozornění.

3.3.3.2 Editace pacienta (add patient)

Editace pacienta je shodná s editací uživatele (viz kapitola 3.1.3.2).

3.3.3.3 Odstranění pacienta (delete patient)

Odstranění pacienta probíhá shodně s odstraněním uživatele (viz. kapitola 3.1.3.3).

3.4 Práce s plány

3.4.1 Popis a priorita

Tato funkce slouží k plánování léčebných procedur pacientům (vytvoření pracovních úkolů zaměstnancům). Umožňuje vytvořit nový plán, editovat již vytvořený plán nebo odstranit vytvořený plán.
Priorita vytvoření plánu je vysoká.

3.4.2 Události a odpovědi

- **Detail** – systém zobrazí podrobné informace o vybraném plánu. V tomto okně nebude možné žádné údaje přepsat. Není – li vybrán žádný plán, bude zobrazeno dialogové okno s upozorněním, že musí být vybrán plán, který má být detailně zobrazen.
- **List** – zobrazí seznam všech přiřazených plánů.
- **Add** – systém zobrazí nevyplněné okno detailu plánu.
- **Edit** – Zobrazí detail vybraného plánu. Jakýkoliv údaj bude možné opravit.
- **Delete** – odstraní vybraný plán (odstranění se neprovádí příznakem, údaje budou po odstranění smazány).
- **Clear** – všechny údaje v položkách detailu budou vyčištěny (pouze v editaci nebo při tvoření, u obyčejného zobrazení detailu není možné data editovat, tudíž ani smazat).
- **Save** – uloží změny provedené v detailu plánu.
- **Unsave** – tato událost se vrátí k poslední uložené verzi okna (byly – li provedeny nějaké změny, nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.4.3 Funkční požadavky

3.4.3.1 Vytvoření plánu (add plan)

Okno vytvoření plánu musí obsahovat všechny potřebné údaje obsažené v tabulkách pro plánování (viz. dokumentace databáze). Po kliknutí na událost „add“ (Přístup je možný buď ze seznamu úkolů, nebo z detailu jakéhokoliv již naplánovaného úkolu. V editaci je pole přidání práce potlačené), se

zobrazí nevyplněné okno plánování. V okně plánování je nutné vyplnit jméno pacienta, kterému bude úkon prováděn a uživatele, který daný úkol bude vykonávat. Dále bude vyplněno pole iterace (údaj, určující denní dobu, kdy je nutno úkon provést – ráno, odpoledne, večer), úkon, který se má vykonat – je možno zadat v jednom plánu větší množství úkonů. Pole materiálu, potřebného pro daný úkon bude vyplněno z databáze, podle toho, jaké množství materiálu je pro daný úkol přiřazeno v číselníku, použitý materiál však bude možno dle potřeby odebrat nebo přidat (je možno zadat množství přidávaného materiálu). Osobní údaje pacienta, jako je diagnóza, indikace, zdravotní péče a jméno ošetřujícího doktora, kterému je úkon plánován, se budou automaticky generovat z databáze. Všechny povinné prvky (prvky, které v databázi nemají povoleno políčko „allow null“) musí být vyplněny, jinak nebude možné nový plán uložit.

3.4.3.2 Editace plánu (edit plan)

Přístup je možný po kliknutí na událost „edit“ buď v detailu daného plánu, nebo v seznamu plánů (pro zobrazení editačního okno je nutno, mít vybrán plán který se má editovat, není-li nic vybráno, uživatel musí být upozorněn). Otevře se okno detailu daného plánu, ve kterém bude povolen zápis do jednotlivých prvků, uživatel tak bude mít možnost jednotlivé údaje upravit. Opět musí být vyplněny veškeré povinné údaje, aby bylo možno editovaný plán uložit.

3.4.3.3 Odstranění plánu (delete plan)

Jakýkoliv plán je možno vymazat z okna seznamu plánů a to po výběru plánu, který má být odstraněn nebo z okna editace daného plánu. Po kliknutí na událost „Delete“ bude uživatel upozorněn, že se chystá vybraný plán odstranit.

3.5 Práce s historií

3.5.1 Popis a priorita

Umožňuje kontrolovat minulé stavy pacientů a procedury, které jim byly v minulosti plánovány.

3.5.2 Události a odpovědi

Detail – systém zobrazí podrobné informace o vybrané historii. V tomto okně nebude možné žádné údaje přepsat. Není – li vybrána žádná historie, bude zobrazeno dialogové okno s upozorněním, že musí být vybránahistorie, která má být detailně zobrazena.
List – zobrazí seznam všech přiřazených historií.
Add – systém zobrazí nevyplněné okno detailu historie.
Edit – Zobrazí detail vybrané historie. Jakýkoliv údaj bude možné opravit.
Clear – všechny údaje v položkách detailu budou vyčištěny (pouze v editaci, u obyčejného zobrazení detailu není možné data editovat, tudíž ani smazat).
Save – uloží změny provedené v detailu historie.
Unsave – tato událost se vrátí k poslední uložené verzi okna (byly – li provedeny nějaké změny, nebudou uloženy).
Filtr – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
Back – umožňuje vrácení aplikace o krok zpět.

Print – umožňuje tisk dané historie.

3.5.3 Funkční požadavky

3.5.3.1 Vytvoření historie

Historie je generována automaticky po ukončení úkolu uživatelem v mobilní aplikaci nebo ukončení hlavní sestrou či administrátorem (jiný uživatel nemá přístup k ukončení plánu) v desktop aplikaci.

3.5.3.2 Editace historie

Viz. editace plánu (kapitola 3.4.3.2)

3.5.3.3 Tisk historie

Přístup k tisku historie má uživatel opět buď ze seznamu historií, označením dané historie a stisknutím tlačítka “tisk” nebo z detailu dané historie, také stisknutím tlačítka “Tisk”. Uživateli se zobrazí klasické POpUp okno pro zadání podrobností o tisku (počet kopií, vybrání tiskárny, nastavení kvality tisku, atd.) a po potvrzení tohoto nastavení bude historie vytištěna.

3.6 Práce s testy a statistikami

Tato funkce bude implementována až v dalších iteracích vývoje systému.

3.6.1 Popis a priorita

3.6.2 Události a odpovědi

3.6.3 Funkční požadavky

3.6.3.1 Generování statistik (generace statistic)

3.6.3.2 Zobrazení statistik (show statistic)

3.6.3.3 Generování testů (generace test)

3.6.3.4 Zobrazení testů (show test)

3.7 Práce se zónami

3.7.1 Popis a priorita

Zóny umožňují spravování oblasti působení agentury a její rozdělení na menší úseky. Do těchto úseků jsou pak pacienti rozděleni dle místa bydliště. Do jednotlivých zón jsou také přiřazováni uživatelé, ti pak mají danou zónu a pacienty v této oblasti na starosti.

3.7.2 Události a odpovědi

- **Detail** – systém zobrazí informace o vybrané zóně a seznam adres, které do zóny patří. V tomto okně není možné údaje upravovat. Pro zobrazení detailu je nutné mít vybránu některou zónu, jinak nebude možné detail zobrazit, aplikace uživatele o problému nevybrání zóny informuje.
- **List** – zobrazí seznam zón.
- **Add** – systém zobrazí prázdné okno zóny, do prvků tohoto okna bude možno zapisovat.
- **Edit** – Zobrazí detail vybrané zóny. Jakýkoliv údaj bude možné opravit.
- **Delete** – odstraní vybranou zónu (odstranění se neprovádí příznakem, údaje budou po odstranění smazány).
- **Clear** – všechny údaje v položkách okna budou vyčištěny (pouze v editaci nebo při tvoření, u obyčejného zobrazení detailu není možné data editovat, tudíž ani smazat).
- **Save** – uloží změny provedené v okně zóny.
- **Unsave** – tato událost se vrátí k poslední uložené verzi okna (byly – li provedeny nějaké změny, nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.7.3 Funkční požadavky

3.7.3.1 Přidání zóny (add zone)

Novou zónu je možné vytvořit opět buď ze seznamu zón, nebo z detailu jakékoliv vybrané zóny. Po kliknutí na událost „add“ bude otevřeno prázdné okno detailu zón, do nějž bude možné zapisovat údaje nové zóny. K zóně bude možné přiřadit adresy, které tato zóna zahrnuje. Aby bylo možné uložit novou zónu do seznamu zón, musí být vyplněny všechny povinné údaje. Nebudou – li povinné údaje vyplněny, bude opět uživatel upozorněn, že musí veškeré povinné údaje vyplnit.

3.7.3.2 Editace zóny (edit zone)

K editaci zón je přístup možný ze seznamu zón nebo z detailu vybrané zóny. Po kliknutí na událost „Edit“ bude zobrazeno okno detailu s možností editace polí detailu. V editaci také bude možné přidávat k editovaným zónám nové náležící adresy nebo ze zóny odebírat adresy, které již do ní patřit nemají. Pro uložené zóny je opět nutné mít vyplněny veškeré povinné údaje pro databázi.

3.7.3.3 Odstranění zóny (delete zone)

Odstranění existující zóny je možné přímo ze seznamu zón, nebo také z detailu vybrané zóny.
K tomuto účelu slouží událost „Delete“. Po kliknutí na tuto událost musí být uživatel upozorněn, že se chystá odstranit zónu a vyzván k potvrzení odstranění. Odstraněním bude zóna smazána z databáze (zóna není mazána příznakem).

3.8 Práce s materiálem

3.8.1 Popis a priorita

Tato funkce slouží k práci s materiálem používaným k provádění pracovních úkonů. Umožňuje zaevidovat nový materiál, editovat již zaevidovaný materiál nebo odstranit materiál, který už se nepoužívá.

3.8.2 Události a odpovědi

- **Detail** – systém zobrazí informace o vybraném materiálu. V tomto okně není možné údaje měnit. Pro zobrazení musí uživatel vybrat jeden z materiálů uvedených v seznamu materiálů, aby bylo možné detail zobrazit. Není - li materiál vybrán, bude uživatel vyzván k vybrání materiálu.
- **List** – zobrazí seznam materiálu.
- **Add** – systém zobrazí nevyplněné okno materiálu s povoleným zápisem do prvků.
- **Edit** – Zobrazí detail vybraného materiálu. Jakýkoliv údaj bude možné opravit.
- **Delete** – odstraní vybraný materiál ze seznamu materiálů (odstranění se neprovádí příznakem, údaje budou po odstranění smazány i z databáze).
- **Clear** – všechny údaje v položkách okna budou vyčištěny (pouze v editaci nebo při tvoření uživatele, u obyčejného zobrazení detailu není možné data editovat, tudíž ani smazat).
- **Save** – uloží změny provedené v okně materiálu.
- **Unsave** – tato událost se vrátí k poslední uložené verzi okna (byly – li provedeny nějaké změny, nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.8.3 Funkční požadavky

3.8.3.1 Přidání materiálu (add material)

Zavedení nového materiálu pracuje stejně, jako zavedení nové zóny do databáze v číselníku zón (viz. kapitola 3.6.3.1). V tomto číselníku nebude možné k materiálu nic přiřazovat, je možné pouze vyplnit údaje nového materiálu a zadat jeho množství na skladě. Materiál bude dále obsahovat popis možnosti použití tohoto materiálu.

3.8.3.2 Editace materiálu (edit material)

Editace materiálu probíhá stejně jako editace zón (viz. kapitola 3.6.3.2)

3.8.3.3 Odstranění materiálu (delete material)

Odstranění materiálu je shodné s odstraněním zóny (viz. kapitola 3.6.3.3)

3.9 Práce se skupinami

3.9.1 Popis a priorita

Práce s tímto číselníkem umožňuje řadit uživatele do určitých pracovních skupin. Například podle směn, na které uživatelé chodí atp.

3.9.2 Události a odpovědi

- **Detail** – systém zobrazí informace o dané skupině. V okně pro detail není možné měnit žádné údaje. K zobrazení detailu je třeba mít vybránu skupinu, která se má zobrazit. Není - li nic vybráno, bude uživatel upozorněn a vyzván k vybrání skupiny.
- **List** – zobrazí seznam všech skupin v databázi.
- **Add** – systém zobrazí nevyplněné okno skupiny s povoleným zápisem do prvků.
- **Edit** – Zobrazí detail vybrané skupiny. Jakýkoliv údaj bude možné opravit.
- **Delete** – odstraní vybranou skupinu ze seznamu skupin (odstranění se neprovádí příznakem, údaje budou po odstranění smazány i z databáze).
- **Clear** – všechny údaje v položkách okna budou vyčištěny (pouze v editaci nebo při tvoření uživatele, u obyčejného zobrazení detailu není možné data editovat, tudíž ani smazat).
- **Save** – uloží změny provedené v okně.
- **Unsave** – tato událost se vrátí k poslední uložené verzi okna (byly – li provedeny nějaké změny, nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.9.3 Funkční požadavky

3.9.3.1 Přidání skupiny (add group)

Vytvoření nové skupiny probíhá stejně jako vytvoření nové zóny (viz. kapitola 3.6.3.1). K nové skupině je možné přiřazovat libovolný počet uživatelů, kteří budou do této skupiny náležet.

3.9.3.2 Editace skupiny (edit group)

Editace vybrané skupiny probíhá stejným způsobem, jako editace zóny (viz. kapitola 3.6.3.2). U editace skupiny je možné odstraňovat libovolného uživatele z dané skupiny nebo k ní přiřazovat nové uživatele.

3.9.3.3 Odstranění skupiny (delet group)

Odstranění skupiny je shodné s odstraněním zóny (viz. kapitola 3.6.3.3).

3.10 Práce s rolemi

3.10.1 Popis a priorita

Řazení uživatelů do rolí umožňuje povolovat nebo zakazovat přístup různým uživatelům do některých částí aplikace. Tato funkce umožňuje zobrazovat nastavení privilegií v rolích zavedených v databázi nebo tyto role editovat a tím jim měnit přiřazení privilegií. Práce s rolemi neumožňuje tvořit nové role nebo stávající role mazat. V systému musí existovat role, která bude mít povolen přístup ke všem funkcím (administrátor) aby nemohlo dojít k situaci, kdy nebude moci k některé funkci přistoupit žádný uživatel.

3.10.2 Události a odpovědi

- **Detail** – systém zobrazí název role a privilegia k ní přiřazené. V tomto okně není možné přiřazení privilegií editovat. K zobrazení role je nutné označit danou roli v seznamu rolí. Není - li nic vybráno, bude uživatel upozorněn a vyzván, aby roli k zobrazení označil.
- **List** – zobrazí seznam všech rolí v databázi.
- **Edit** – Zobrazí detail vybrané role. Zde je možné privilegia dané roli libovolně přiřazovat nebo odebírat.
- **Save** – uloží změny provedené v okně.
- **Unsave** – tato událost se vrátí k poslední uložené verzi okna (byly – li provedeny nějaké změny, nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.10.3 Funkční požadavky

3.10.3.1 Editace role (edit role)

Toto okno se zobrazí po kliknutí na událost „Edit“ z okna seznamu rolí nebo v detailu již vybrané role. Poté bude zobrazeno okno detailu role a bude uživateli umožněno přiřazovat privilegia editované roli nebo této roli její privilegia odebírat.

3.11 Práce s úkoly

3.11.1 Popis a priorita

Funkce umožňuje práci s číselníky spravujícími úkoly, které je možno plánovat pacientům. Je možno vkládat do číselníku nové úkoly, editovat existující, nebo odstraňovat některé úkoly z číselníku (například ty úkoly, které se již nepoužívají).

3.11.2 Události a odpovědi

- **Detail** – Zobrazí detailní údaje o vybraném úkolu. Zde není možno údaje editovat.
- **List** – zobrazí seznam úkolů uložených v číselníku úkolů.
- **Edit** – Zobrazí detail vybraného úkolu s možností měnit údaje v číselníku.

- **Delete** – odstraní vybranou skupinu ze seznamu skupin (odstranění se neprovádí příznakem, údaje budou po odstranění smazány i z databáze).
- **Save** – uloží provedené změny.
- **Unsave** – tato událost se vrátí k poslední uložené verzi okna (změny nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.11.3 Funkční požadavky

3.11.3.1 Přidání nového úkolu (add new work)

Vytvoření nového úkolu je shodné s vytvořením nové zóny (viz. kapitola 3.6.3.1). K úkolům je také možno přiřazovat materiál a jeho množství, které je pro splnění úkolu potřebné.

3.11.3.2 Editace úkolu (edit work)

Editace úkolu je stejná, jako editace zóny (viz. kapitola 3.6.3.2). Editace úkolu také umožňuje přidávat, či odebírat materiál, používaný k úkolu či měnit množství použitého materiálu.

3.11.3.3 Odstranění úkolu (delete work)

viz. kapitola 3.6.3.3

3.12 Práce s diagnózami

3.12.1 Popis a priorita

Funkce slouží k práci s diagnózami. Do číselníku je možno diagnózy přidávat, editovat nebo odstraňovat.

3.12.2 Události a odpovědi

- **Detail** – Zobrazí detail vybrané diagnózy. Zde není možno údaje editovat.
- **List** – zobrazí seznam diagnóz uložených v číselníku úkolů.
- **Edit** – Zobrazí detail vybrané diagnózy s možností měnit údaje v číselníku.
- **Save** – uloží provedené změny.
- **Unsave** – vrátí aplikaci k poslední uložené verzi (změny nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.12.3 Funkční požadavky

3.12.3.1 Přidání diagnózy (add diagnosis)

viz. kapitola 3.6.3.1

3.12.3.2 Editace diagnózy (edit diagnosis)

viz. kapitola 3.6.3.2

3.12.3.3 Odstranění diagnózy (delete diagnosis)

viz. kapitola 3.6.3.1

3.13 Práce s dietami

3.13.1 Popis a priorita

Práce s dietami umožňuje vkládat do číselníku nové diety a editovat nebo odstraňovat již vytvořené diety.

3.13.2 Události a odpovědi

- **Detail** – Zobrazí detail vybrané diety. V tomto okně není umožněno žádné údaje editovat.
- **List** – zobrazí seznam všech diet v číselníku.
- **Edit** – Zobrazí okno detailu dané diety. V tomto okně mohou být údaje editovány. Nikdy nelze editovat pořadové číslo položky (ID).
- **Save** – uloží provedené změny.
- **Unsave** – vrátí aplikaci k poslední uložené verzi (změny nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.13.3 Funkční požadavky

3.13.3.1 Přidání diety (add diet)

viz. kapitola 3.6.3.1

3.13.3.2 Editace diety (edit diet)

viz. kapitola 3.6.3.2

3.13.3.3 Odstranění diety (delete diet)

viz. kapitola 3.6.3.3

3.14 Práce se zdravotní péčí

3.14.1 Popis a priorita

Tato funkce umožňuje spravovat číselník s údaji o zdravotní péči. Tyto údaje je možno editovat, dále je možno vytvářet novou zdravotní péči nebo odstraňovat typ zdravotní péče, který je již v databázi uložen.

3.14.2 Události a odpovědi

- **Detail** – Zobrazí detail dané zdravotní péče. Zdravotní péče, která má být zobrazena, musí být označena, nebude – li tak učiněno, bude uživatel upozorněn, že nevybral žádnou zdravotní péči a vyzván k jejímu označení. V tomto okně není umožněno žádné údaje editovat.
- **List** – zobrazí seznam všech typů zdravotních péčí uložených v databázi tohoto číselníku.
- **Edit** – Zobrazí okno detailu daného typu zdravotní péče. V tomto okně mohou být údaje editovány. Nikdy nelze editovat pořadové číslo položky (ID).
- **Save** – uloží provedené změny.
- **Unsave** – vrátí aplikaci k poslední uložené verzi (změny nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.14.3 Funkční požadavky

3.14.3.1 Přidání zdravotní péče (add health care)

viz. kapitola 3.6.3.1

3.14.3.2 Editace zdravotní péče (edit healh care)

viz. kapitola 3.6.3.2

3.14.3.3 Odstranění zdravotní péče (delete health care)

viz. kapitola 3.6.3.3

3.15 Práce s indikacemi

3.15.1 Popis a priorita

Funkce slouží k správě číselníku indikací. V tomto číselníku je umožněno vytvářet nové indikace, editovat a odstraňovat již vytvořené indikace.

3.15.2 Události a odpovědi

- **Detail** – Zobrazí detail vybrané indikace. Není – li žádná indikace vybrána, bude uživatel upozorněn. Zde není možno žádné údaje editovat.
- **List** – zobrazí seznam všech indikací v číselníku.
- **Edit** – Zobrazí detailu dané indikace. V tomto okně mohou být údaje o vybrané indikaci editovány. Nikdy nelze editovat pořadové číslo položky (ID).
- **Save** – uloží provedené změny.
- **Unsave** – vrátí aplikaci k poslední uložené verzi (změny nebudou uloženy).
- **Filtr** – umožňuje uživateli vyhledávání v aplikaci pomocí zadaných parametrů.
- **Back** – umožňuje vrácení aplikace o krok zpět.

3.15.3 Funkční požadavky

3.15.3.1 Přidání indikace(add health care)

viz. kapitola 3.6.3.1

3.15.3.2 Editace indikace (edit healh care)

viz. kapitola 3.6.3.2

3.15.3.3 Odstranění indikace (delete health care)

viz. kapitola 3.6.3.3

4 Požadavky na vnější rozhraní

Systém je nucen bezproblémově komunikovat se serverovým klientem, z něhož systém získává veškerá data, se kterými bude pracovat. Tento klient bude uložen buď na PC společně s PC klientem administrátora, nebo na svém vlastním PC. Dále je třeba zajistit správnou funkci GUI (grafical user interface) – grafické uživatelské rozhraní a komunikace mezi databázovým systémem a PC klienty.

4.1 Uživatelská rozhraní

Uživatelské rozhraní bude pouze jedno, ovšem pro uživatele s jiným oprávněním než administrátorským budou skryty funkce, ke kterým nebude mít přihlášený uživatel přístup.

- GUI bude obsahovat standardní ovládací prvky definované .NET frameworkem 3.5.
- Každá obrazovka musí nutně obsahovat tlačítko pro vrácení do seznamu aktuálního okna (například seznamu pacientů, jsem – li v okně pacientů) a tlačítko pro odhlášení uživatele.
- Každá zobrazená ***chybová zpráva*** musí obsahovat stručný popis chyby, která nastala (např. uživatelské jméno nebo heslo je neplatné).

- Každé zobrazené upozornění bude obsahovat stručný popis chyby, které se uživatel dopustil a radu, jak má dále postupovat (např. „Nebyl vybrán uživatel, který má být editován. Vyberte nejdříve uživatele.“).
- Každá zobrazená informační zpráva bude obsahovat pouze informaci, kterou má oznámit (např. „změny byly uloženy.“).

Aplikace bude lokalizována do českého jazyka. Lokalizovány musí být veškeré „labely“ popisky tlačítek, toolstripbuttony atd. (veškerý text popisující tlačítka, textboxy nebo jiný text viditelný v uživatelském rozhraní).

4.2 Hardwarová rozhraní

Systém nepočítá s žádnými hardwarovými rozhraní.

4.3 Softwarová rozhraní

Systém bude propojen s databázovým serverem, tento server bude spravovat, uchovávat a poskytovat veškerá data, se kterými systém pracuje (viz. dokumentace databáze). Systém bude s databází propojen buď síťovým kabelem, nebo budou se systémem uloženy na stejném PC. Systém bude k datům přistupovat pomocí technologie LINQ. Databázový server poběží na operačním systému Windows XP/Vista a to z důvodu, že chceme použít pro vývoj databáze MS SQL server a pro vývoj systému .NET framework 3.5. Data, která si databázový server se systémem vyměňují, mají účel osobních dat pacientů či zaměstnanců, ale také data obsahující informace o prováděných úkonech, použitých materiálech, typech zdravotní péče atd.

4.4 Synchronizační/Komunikační rozhraní

Pro komunikaci mobilního klienta s databází bude použit standard pro lokální bezdrátové sítě Wi-Fi, využíván bude protokol TCP/IP.

5 Další parametrické požadavky

5.1 Výkonnostní požadavky

Aplikace nebude pracovat v real – time režimu, data budou dodávána do aplikace v kolekcích, to znamená, že nebude třeba stále přistupovat k databázi, ale data se stáhnou pouze jedenkrát a zapíší do databáze až při uložení. Aplikace tedy nebude náročná na výkon. Odezva systému musí být kratší než významně dlouhá (aby odezva v uživateli nevyvolala pocit nežádoucí nebo zbytečné prodlevy). Systém poběží na PC s pamětí alespoň 2GB RAM, windows XP či Vista. Dotazy na databázi budou vyřízeny nejpozději do 5 sekund pro případ 1000 záznamů v databázi a 2 simultánní přístupy.

5.2 Bezpečnostní požadavky

- **BEZ-1** pro přístup do systému se musí každý uživatel přihlásit pod svým uživatelským jménem.
- **BEZ-2** každý uživatel má přidělenu roli podlé které smí přistupovat jen k funkcím, které má povoleny.
- **BEZ-3** systém bude vyžadovat značnou míru fyzického zabezpečení proti úniku dat.
- **BEZ-4** systém vyžaduje softwarové zabezpečení dostupnýma prostředky.

5.3 Kvalitativní parametry

Z pohledu vývojáře:

- Udržovatelnost
  - Systém musí být vhodně popsán komentáři, aby v něm byla jednodušší orientace.
  - K systému musí být sepsána podrobná dokumentace implementace.
- Znovupoužitelnost
  - Systém pro správu a plánování biomedicínských úkonů v agentuře pro domácí péči musí být navržen tak, aby byl použitelný i pro jinou správu dat a pracovních úkolů jen s menšími úpravami.

Z pohledu uživatele:

- Spolehlivost
  - systém by měl úspěšně dokončit 95% operací, které bude provádět.
- Odolnost
  - systém musí být odolný proti špatně zadaným vstupům uživatelem, překlepům a dalším nevynuceným chybám.
  - Systém musí být také odolný proti nevyplnění povinného údaje.
- Integrita
  - přístup do systému musí být zabezpečen heslem.
  - K datům mají přístup pouze ti zaměstnanci, kteří jsou k tomu oprávněni přidělením role s tímto přístupem.
- Použitelnost
  - Systém musí být intuitivní natolik, aby s ním byl uživatel, který ho vidí poprvé, schopen bez jakýchkoliv větších problémů pracovat.

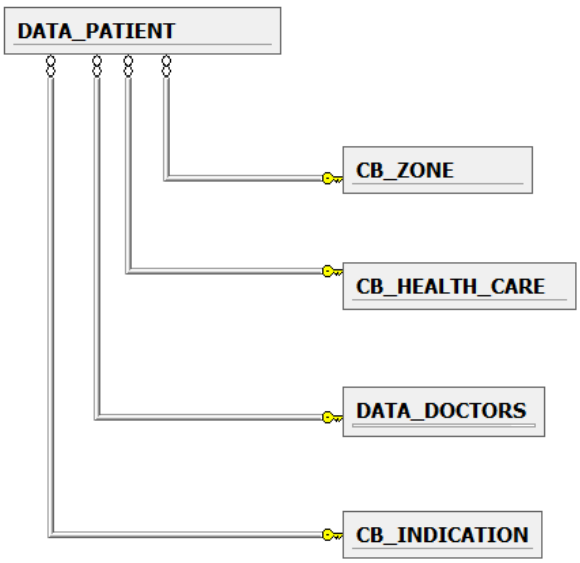
Dodatek: Analytické modely

Viz. Analýza systému (dokument „Diplomová práce“ kapitola 7.3.3).

Příloha II Návrh databáze

Tabulky budou navrženy tak, aby se všechna data vyskytovala pouze v jedné tabulce, do dalších potřebných tabulek budou pro snadnější změnu dat předávány unikátním primárním klíčem (primárním klíčem nesmí být žádná data, která by mohla být editována, nejlépe pro primární klíč vytvořit speciální kód).

Relace tabulky Data_Pacient:



Data tabulky a jejich datové typy:

Název položky	Datový typ	PK či FK
Id pacienta	smallint	PK
Jméno pacienta	Varchar(20)	
Příjmení pacienta	Varchar(20)	
Datum narození	date	
Rodné číslo	varchar(15)	
Adresa	Varchar (50)	
ID Zóny	smallint	FK
Číslo pojišťovny	Smallint	
ID Ošetřujícího lékaře	smallint	FK
ID zdravotní péče	smallint	FK
Indikace	smallint	
Kontakt	Varchar(50)	
Kontaktní číslo	Vardar(14)	
Datum od	Date	
Datum do	Date	
Poznámky	Varchar (100)	
Odstraněn	bool	

Relace tabulky Data_Doctor:

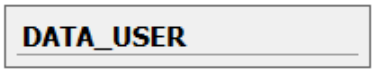
Tabulka obsahuje jen data, nepoužívá data žádné jiné tabulky.

Bude obsahovat vlastní primární klíč, díky kterému bude použita v jiných tabulkách.

Data tabulky a jejich datové typy:

Název položky	Datový typ	PK či FK
ID doktora	smallint	PK
Jméno doktora	Varchar(20)	
Příjmení doktora	Varchar(20)	
Specializace	Varchar(20)	
Telefonní číslo	Varchar(15)	
Heslo	Varchar(20)	
Adresa	Varchar(50)	
Přihlašovací jméno	Varchar(20)	
Odstraněn	bool	

Relace tabulky Data_User:

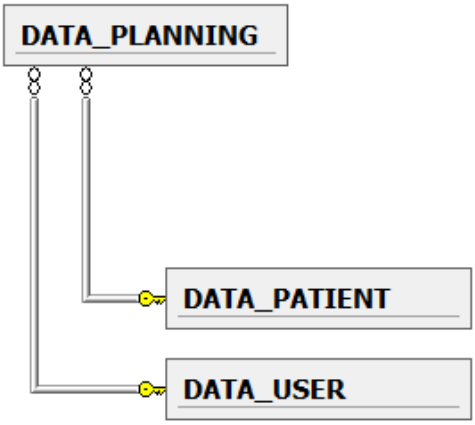


Tato tabulka bude stejným případem jako tabulka data_Doctors, nemá žádné používané relace.

Data tabulky a jejich datové typy:

Název položky	Datový typ	PK či FK
ID uživatele	smalint	PK
Jméno uživatele	Varchar(20)	
Příjmení uživatele	Varchar(20)	
Heslo	Varchar(100)	
Adresa	Varchar(50)	
Telefonní číslo	Varchar(15)	
Přihlašovací jméno	Varchar(20)	
Odstraněn	bool	

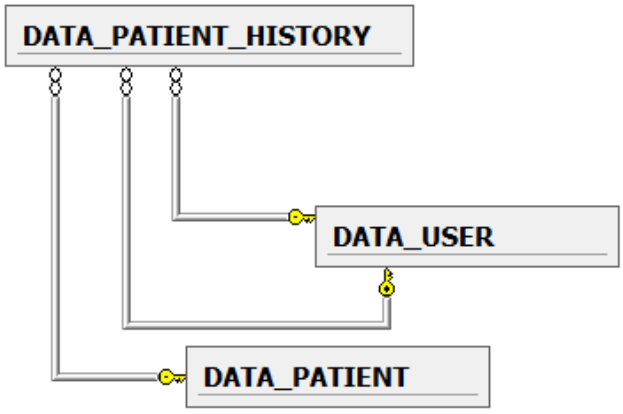
Relace tabulky Data_Planning:



Data tabulky a jejich datové typy:

Název položky	Datový typ	PK či FK
ID plánu	int	PK
ID pacienta	smallint	FK
ID uživatele	smallint	FK
Datum a čas	datetime	
Iterace	tinyint	

Relace tabulky Data_Patient_History:

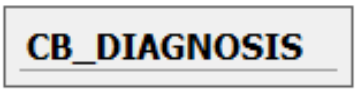


Data tabulky a jejich datové typy:

Název položky	Datové typy	PK či FK
ID historie	int	PK
ID plánu	int	FK
ID pacienta	smallint	FK

ID uživatele	smallint	FK
Původní ID uživatele	smallint	FK
Datum a čas	datetime	
Původní datum a čas	Datetime	
Aktuální stav	Varchar(50)	
Iterace	tinyint	
Popis	Varchar(100)	

Relace tabulky CB_Diagnosis:



Data tabulky a jejich datové typy:

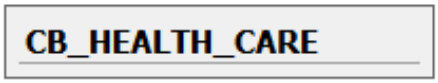
Název položky	Datový typ	PK či FK
ID diagnózy	smallint	PK
Název diagnózy	Varchar(20)	
Kódový název diagnózy	Varchar(10)	
Pojišťovní kód diagnózy	Varchar(10)	
Popis	Varchar(100)	

Relace tabulky CB_Zone:

Data tabulky a jejich datové typy:

Název položky	Datový typ	PK či FK
ID zóny	smallint	ID
Název zóny	Varchar(50)	
Popis	Varchar(100)	

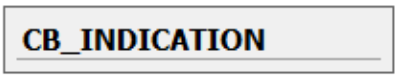
Relace tabulky CB_Health_Care:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID péče	smallint	PK
Název péče	Varchar(20)	
Pojišťovní kód péče	Varchar(10)	
Popis	Varchar(100)	

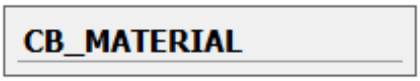
Relace tabulky CB_Indication:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID indikace	smallint	PK
Název indikace	Varchar(20)	
Pojišťovní kód indikace	Varchar(10)	
Popis	Varchar(100)	

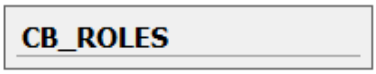
Relace tabulky CB_Material:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID materiálu	smallint	PK
Název materiálu	Varchar(20)	
Množství	smallint	
Jednotka množství	Varchar(10)	
Pojišťovní kód materiálu	Varchar(10)	
Popis	Varchar(50)	

Relace tabulky CB_Roles:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID role	Smallint	PK
Název role	Varchar(20)	

Právo vytvořit pacienta	Bit	
Právo editovat pacienta	Bit	
Právo odstranit pacienta	Bit	
Právo zobrazit pacienta	Bit	
Právo vytvořit uživatele	Bit	
Právo editovat uživatele	Bit	
Právo odstranit uživatele	Bit	
Právo zobrazit uživatele	Bit	
Právo vytvořit plán	Bit	
Právo editovat plán	Bit	
Právo přiřadit plán	Bit	
Právo odstranit plán	Bit	
Právo vytvořit číselník	Bit	
Právo zobrazit číselník	Bit	
Právo editovat číselník	Bit	
Právo odstranit položku čís.	Bit	
Právo generovat test	Bit	
Právo editovat test	Bit	
Právo zobrazit test	Bit	
Právo odstranit test	Bit	
Právo generovat statistiku	Bit	
Právo editovat statistiku	Bit	
Právo odstranit statistiku	Bit	
Právo zobrazit statistiku	bit	

Relace tabulky CB_Work:

Data tabulky a jejich datové typy:

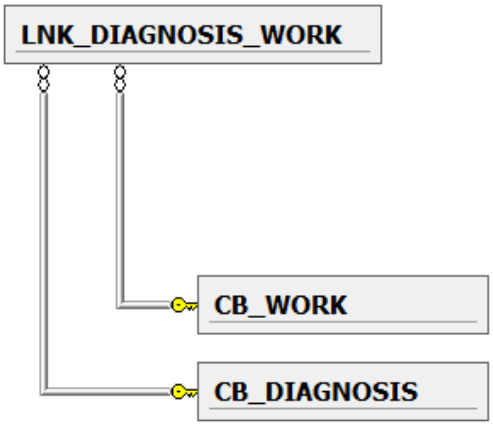
Název tabulky	Datový typ	PK či FK
ID úkonu	smallint	PK
Kódový název úkonu	Varchar(10)	
Název úkonu	Varchar(20)	
Pojišťovní kód úkonu	Varchar(10)	
Popis	Varchar(100)	

Relace tabulky CB_Zone:

Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID zóny	smallint	PK
Název zóny	Varchar(50)	
Popis	Varchar(100)	

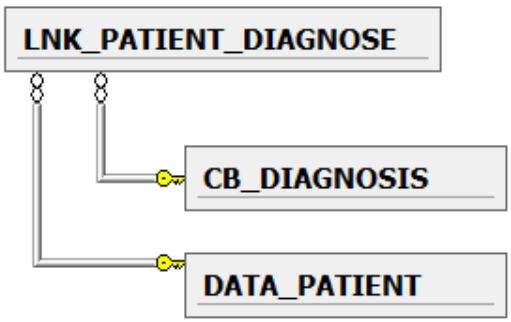
Relace tabulky LNK_Diagnosis_Work:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID položky	int	PK
ID diagnózy	smallint	FK
ID úkonu	smallint	FK

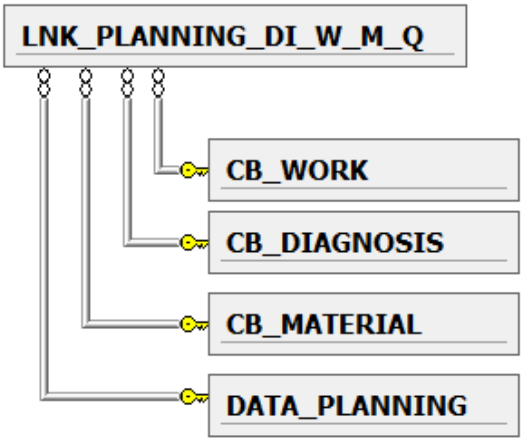
Relace tabulky LNK_Patient_Diagnose:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID položky	smallint	PK
ID diagnózy	smallint	FK
ID pacienta	smallint	FK

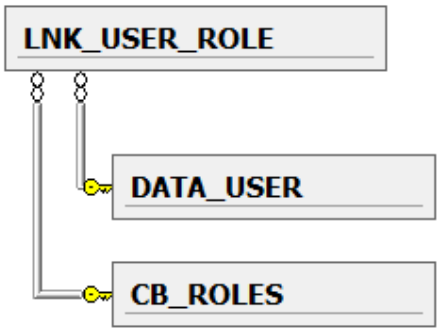
Relace tabulky LNK_Planning_DI_W_M_Q:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID položky	smallint	PK
ID pacienta	smallint	FK
ID diagnózy	smallint	FK
ID práce	Smallint	FK
ID materiálu	smallint	FK
Množství	smallint	

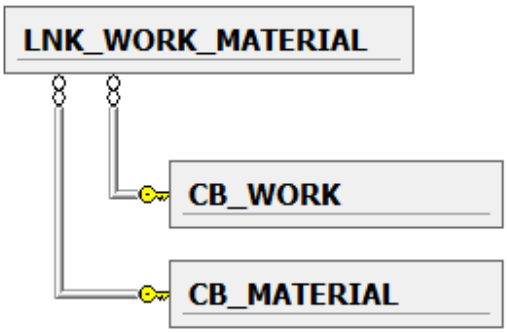
Relace tabulky LNK_User_Role:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID položky	int	PK
ID role	smallint	FK
ID uživatele	smallint	FK

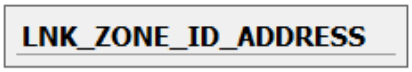
Relace tabulky LNK_Work_Material:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID položky	smallint	PK
ID úkonu	smallint	FK
ID materiálu	smallint	FK

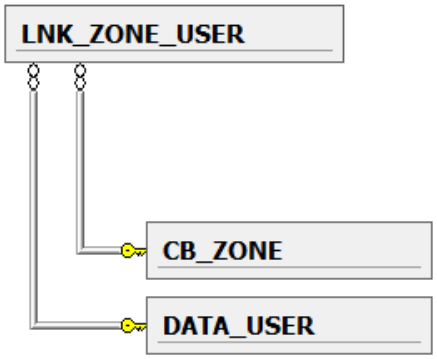
Relace tabulky LNK_Zone_ID_Address:



Data tabulky a jejich datové typy:

Název tabulky	Datový typ	PK či FK
ID položky	int	PK
ID úkonu	smallint	FK
ID materiálu	smallint	FK

Relace LNK_Zone_User:

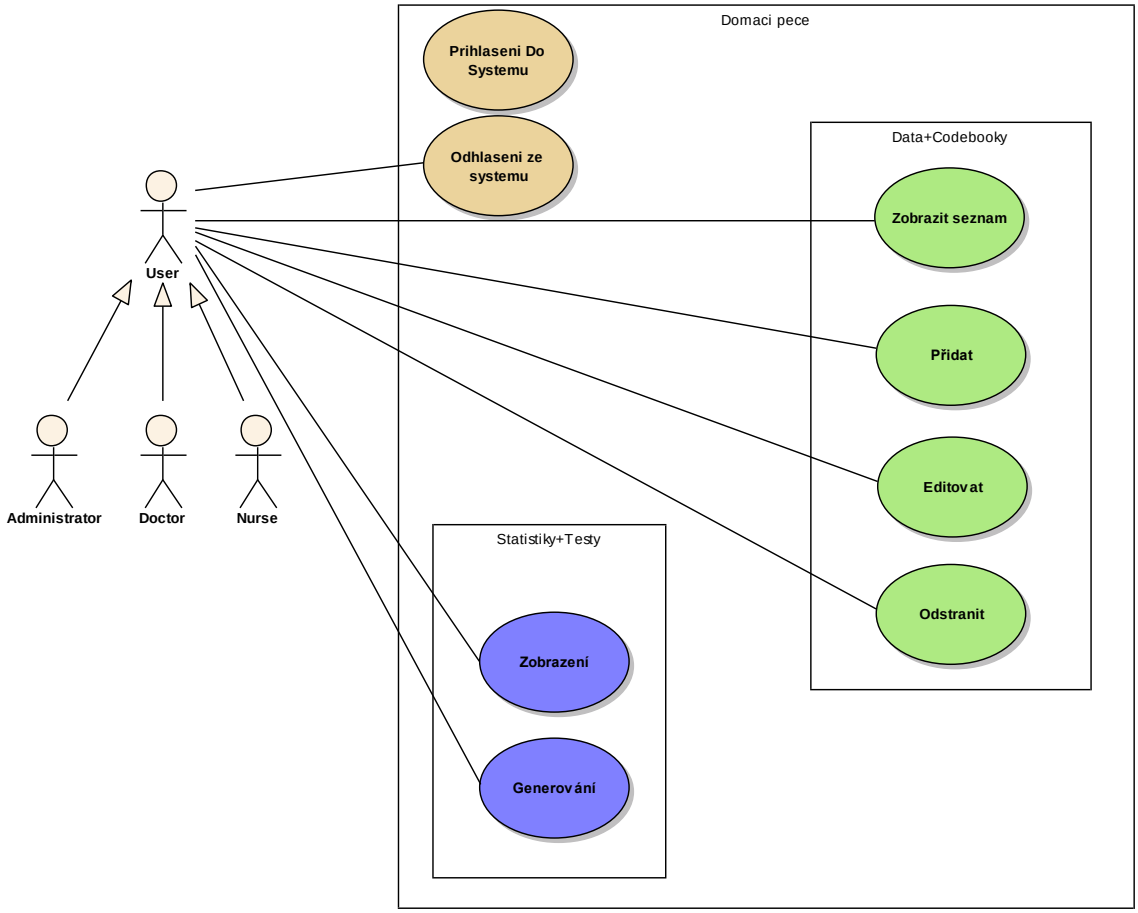


Data tabulky a jejich datové typy:

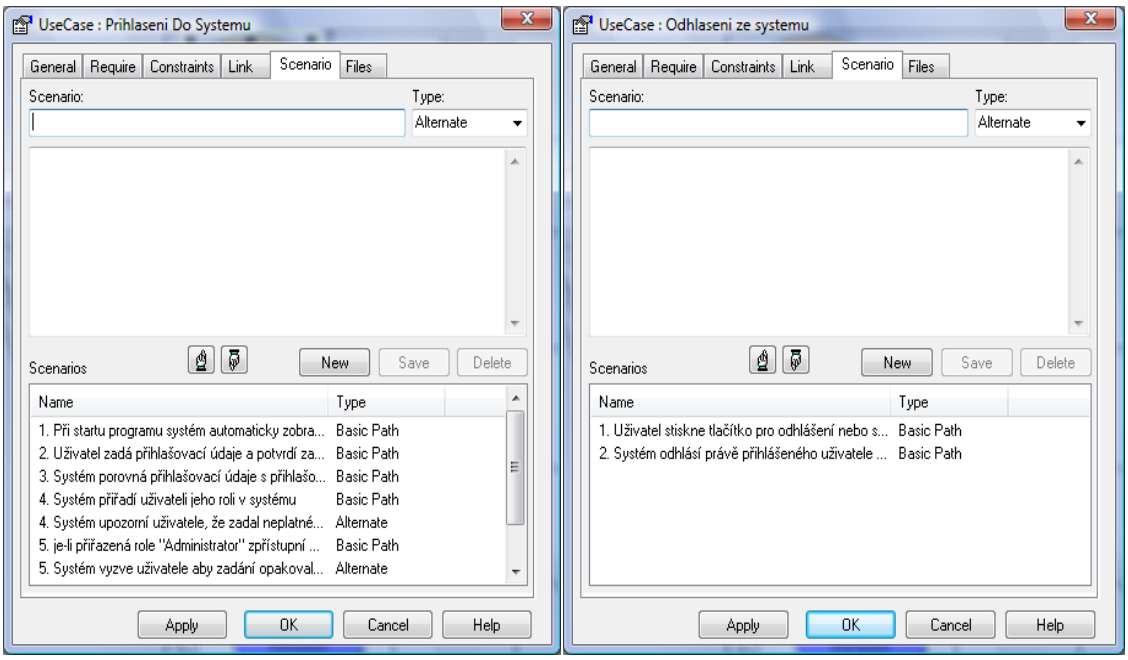
Název tabulky	Datový typ	PK či FK
ID položky	int	PK
ID zóny	smallint	FK
ID uživatele	smallint	FK

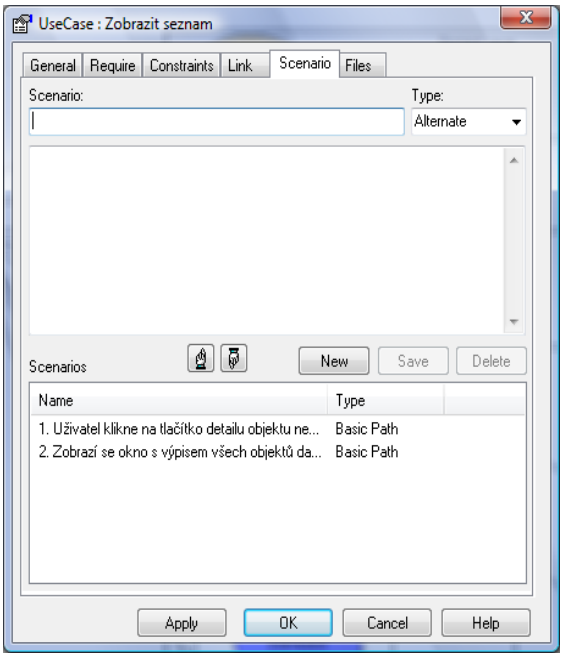
Příloha III Analytické modely

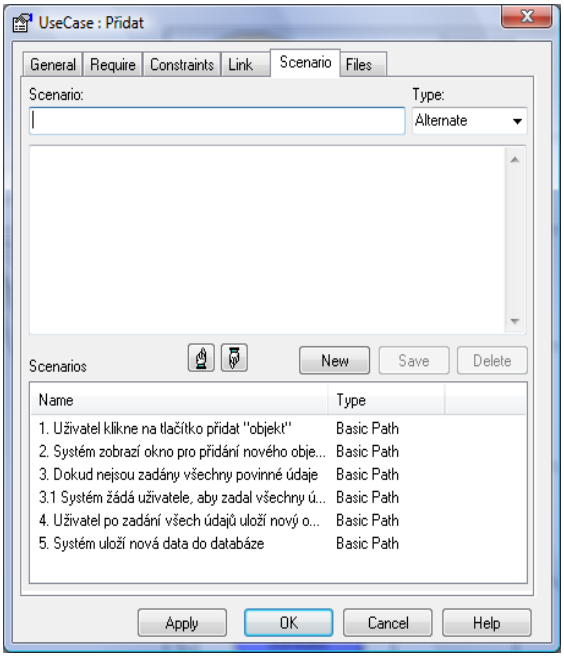
Use Case diagram:

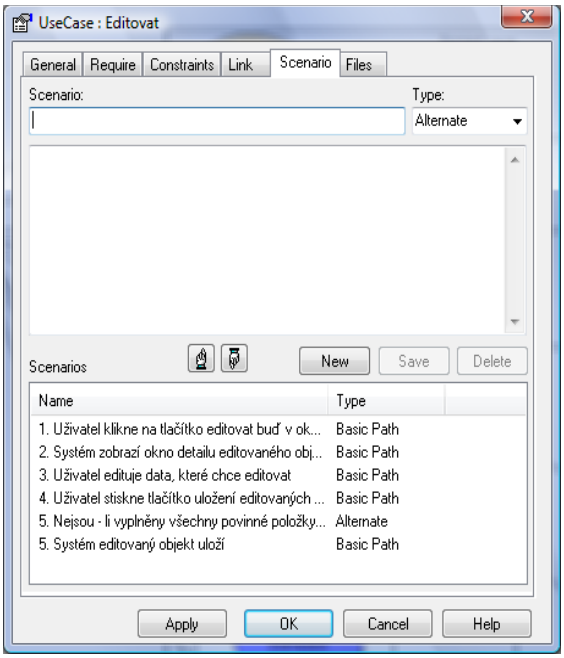


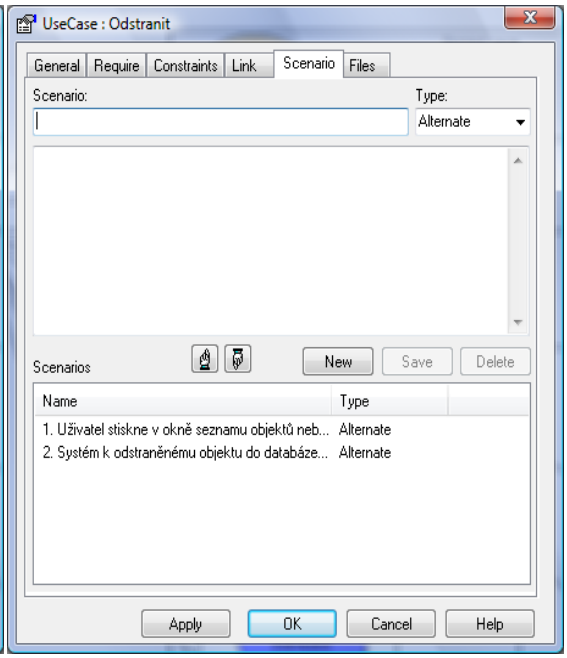
Scénáře jednotlivých případů užití:



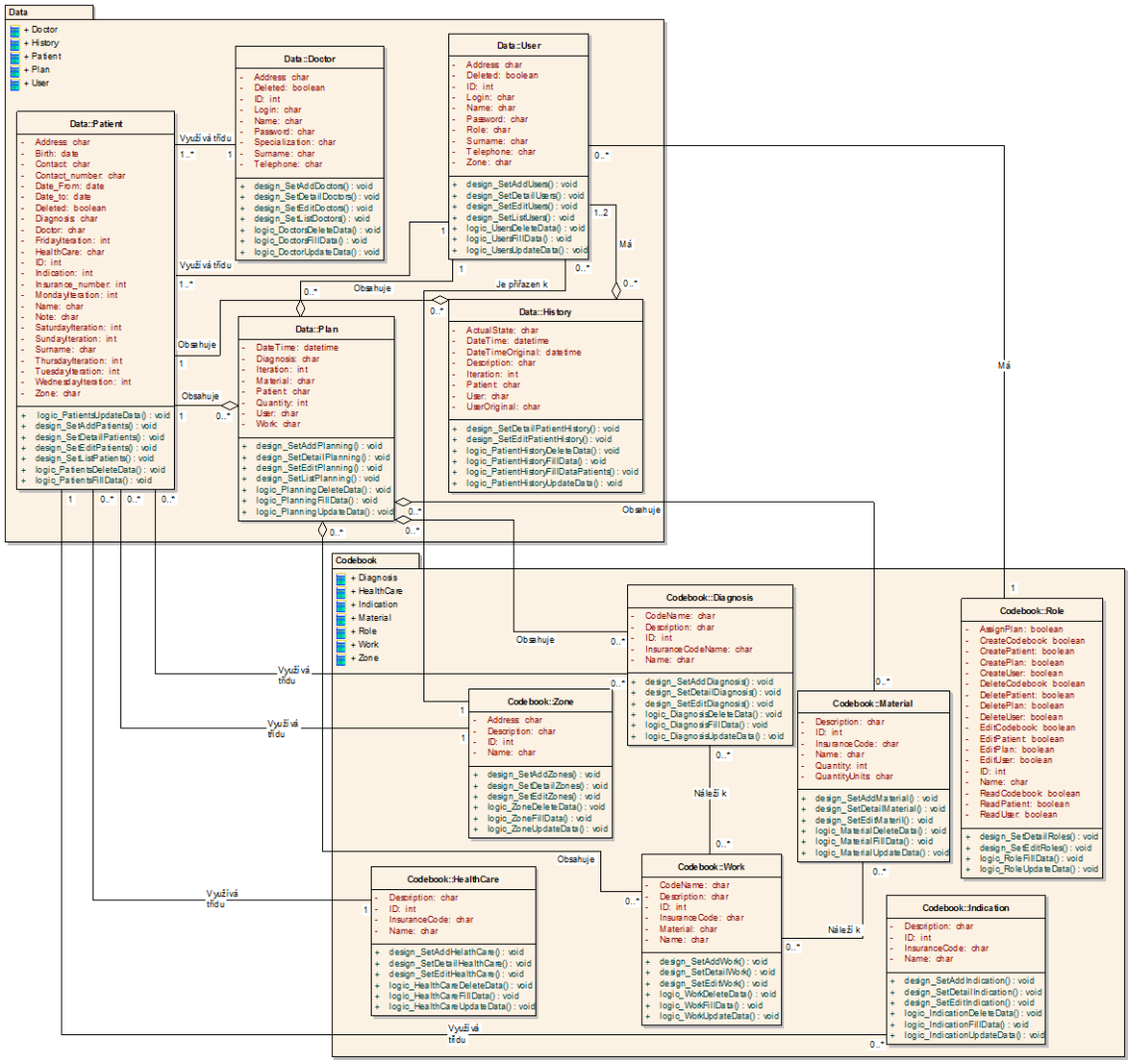

UseCase : Zobrazit seznam
General
Require
Constraints
Link
Scenario
Files
Scenario:
Type:
Alternate
Scenarios
New
Save
Delete
Name
Type
1. Uživatel klikne na tlačítko detailu objektu ne...
Basic Path
2. Zobrazí se okno s výpisem všech objektů da...
Basic Path
Apply
OK
Cancel
Help


UseCase : Přidat
General
Require
Constraints
Link
Scenario
Files
Scenario:
Type:
Alternate
Scenarios
New
Save
Delete
Name
Type
1. Uživatel klikne na tlačítko přidat "objekt"
Basic Path
2. Systém zobrazí okno pro přidání nového obje...
Basic Path
3. Dokud nejsou zadány všechny povinné údaje
Basic Path
3.1 Systém žádá uživatele, aby zadal všechny ú...
Basic Path
4. Uživatel po zadání všech údajů uloží nový o...
Basic Path
5. Systém uloží nová data do databáze
Basic Path
Apply
OK
Cancel
Help

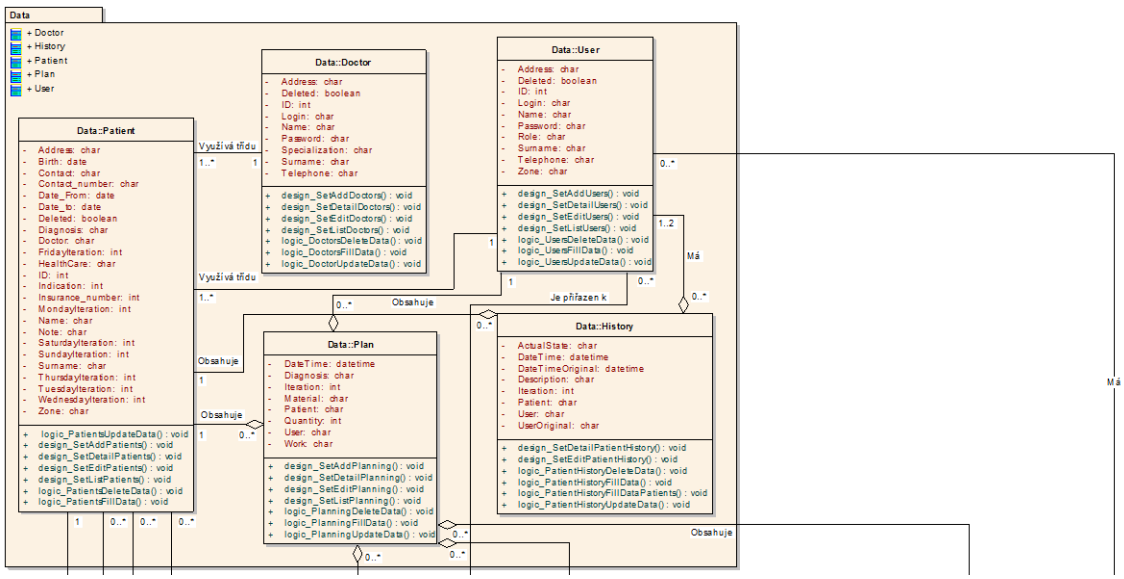

UseCase : Editovat
General
Require
Constraints
Link
Scenario
Files
Scenario:
Type:
Alternate
Scenarios
New
Save
Delete
Name
Type
1. Uživatel klikne na tlačítko editovat buď v ok...
Basic Path
2. Systém zobrazí okno detailu editovaného obj...
Basic Path
3. Uživatel edituje data, které chce editovat
Basic Path
4. Uživatel stiskne tlačítko uložení editovaných ...
Basic Path
5. Nejsou - li vyplněny všechny povinné položky...
Alternate
5. Systém editovaný objekt uloží
Basic Path
Apply
OK
Cancel
Help


UseCase : Odstranit
General
Require
Constraints
Link
Scenario
Files
Scenario:
Type:
Alternate
Scenarios
New
Save
Delete
Name
Type
1. Uživatel stiskne v okně seznamu objektů neb...
Alternate
2. Systém k odstraněnému objektu do databáze...
Alternate
Apply
OK
Cancel
Help

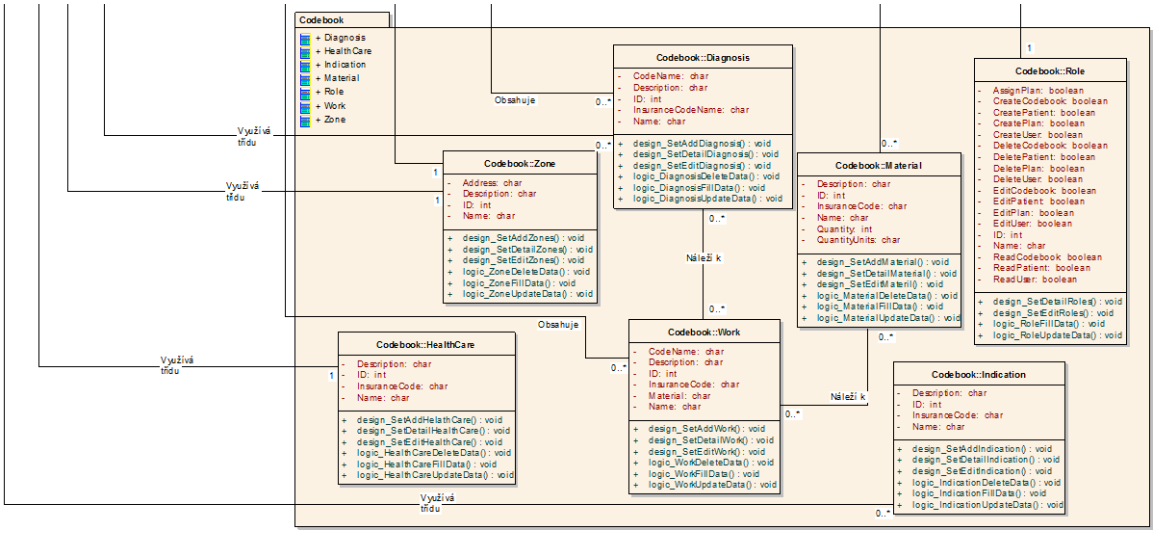
Class diagram:



- **Balíček Data**

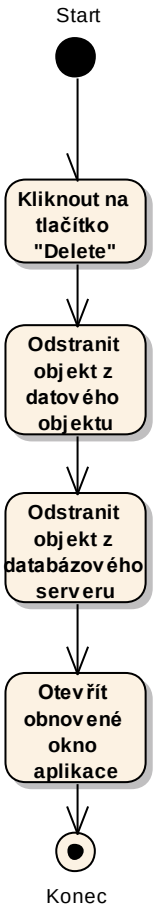


- **Balíček Codebook**

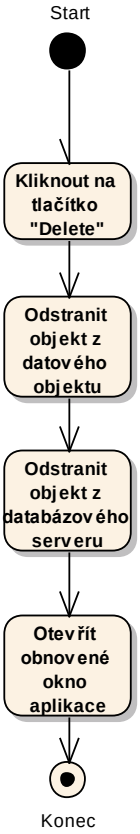


Activity diagramy:

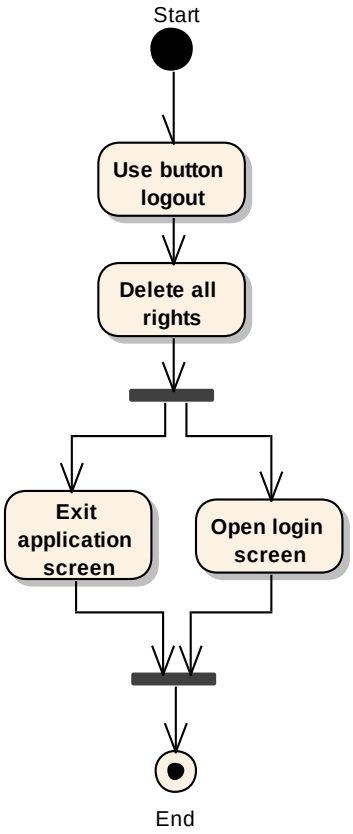
- **Odstranit object**



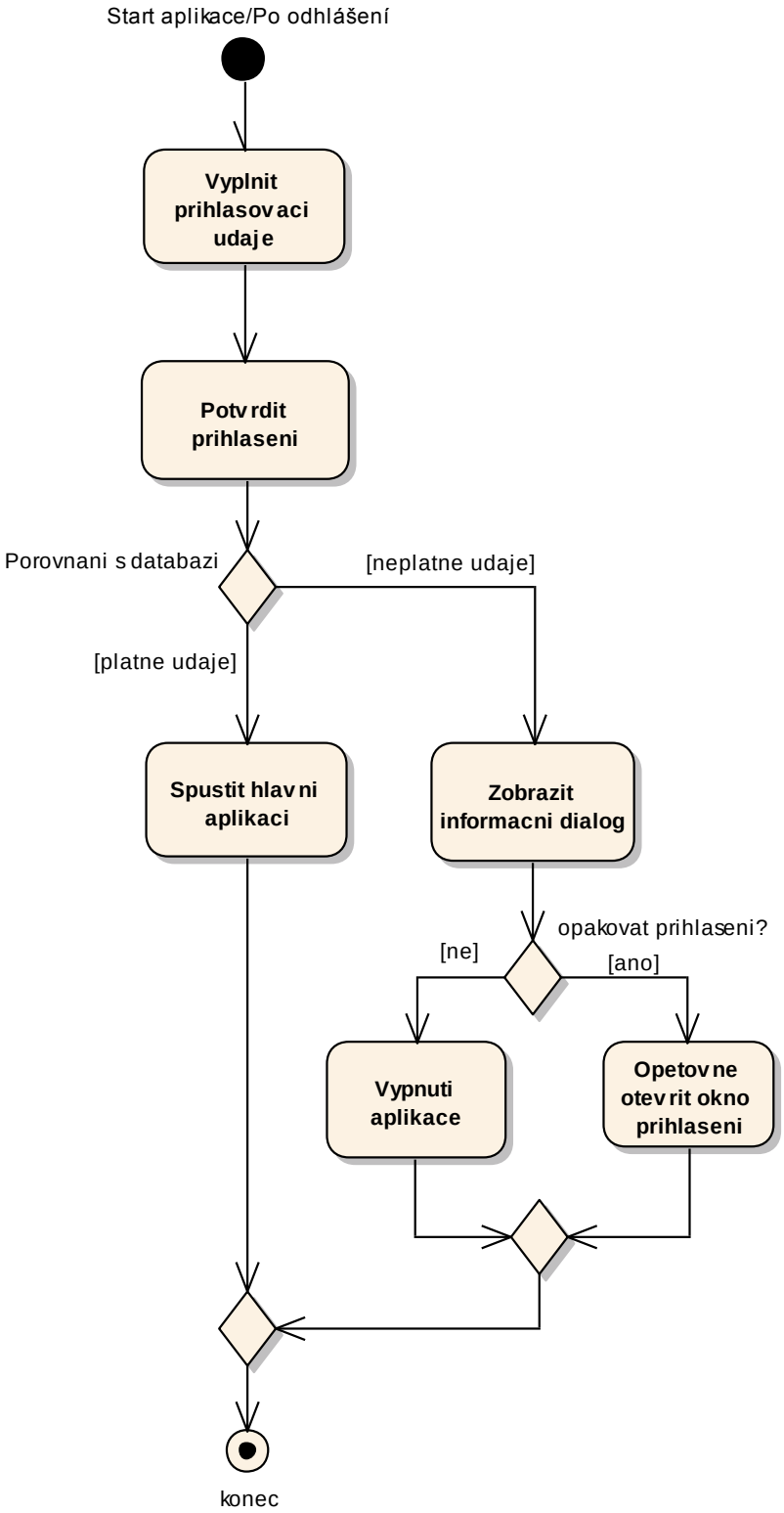
- **Odstranit objekt přidáním atributu**



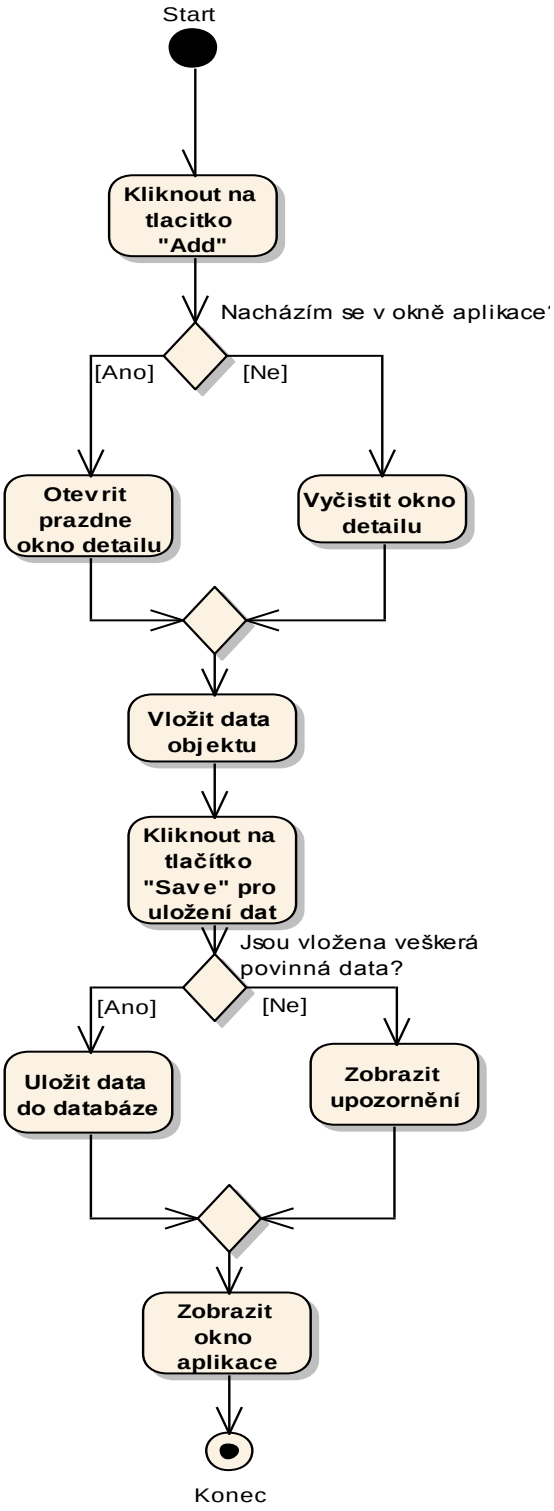
- **Odhlášení**



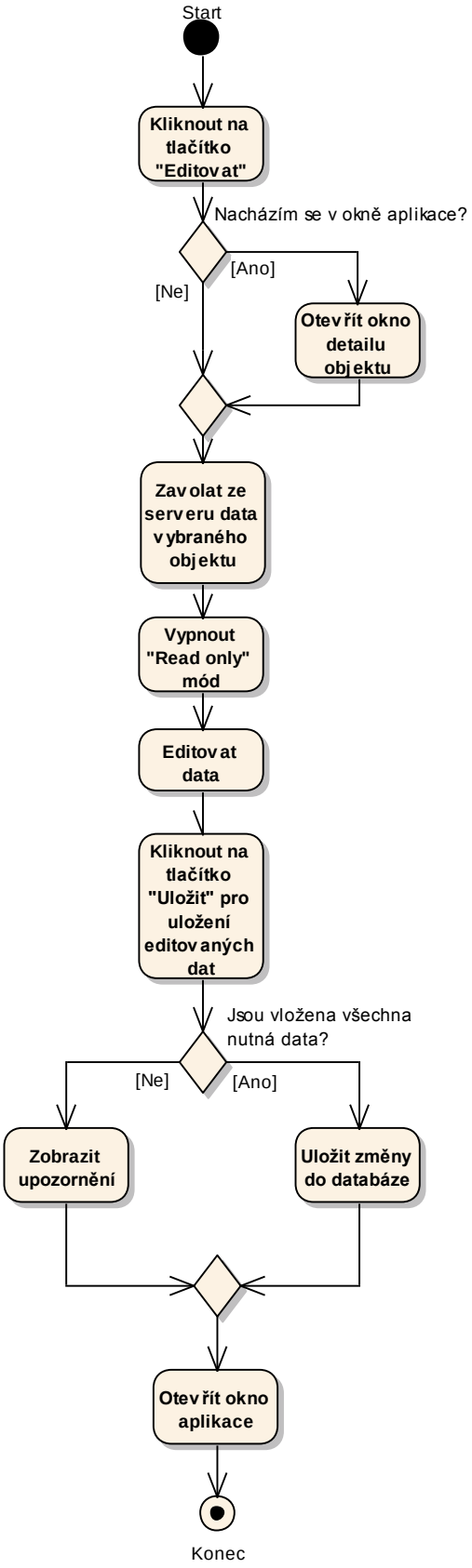
- **Přihlášení**


Start aplikace/Po odhlášení
Vyplnit prihlasovaci udaje
Potvrdit prihlaseni
Porovnani s databazi
[neplatne udaje]
[platne udaje]
Spustit hlavni aplikaci
Zobrazit informacni dialog
opakovat prihlaseni?
[ne]
[ano]
Vypnuti aplikace
Opetovne otevrit okno prihlaseni
konec

- **Přidat objekt**

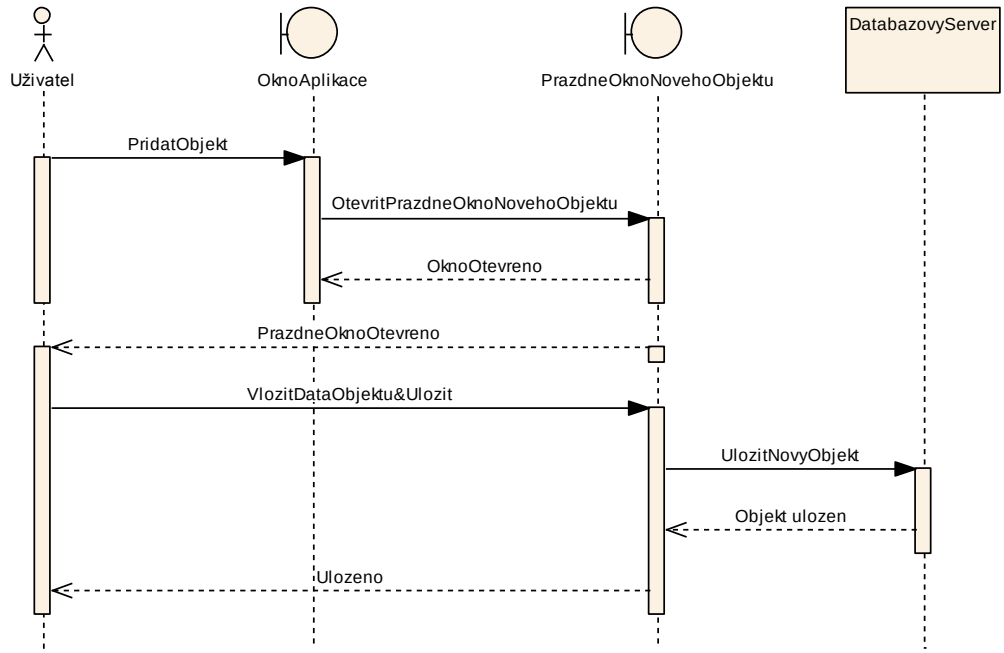


- **Editovat object**

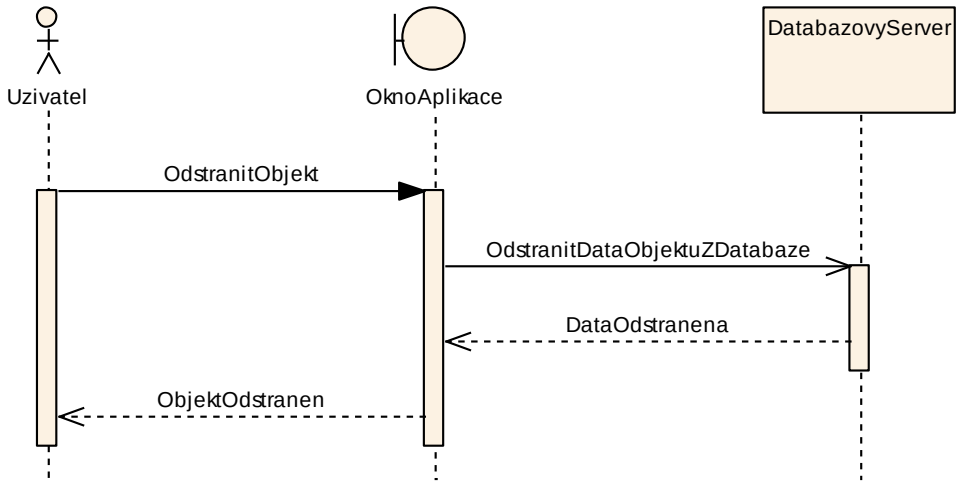


Sekvenční diagramy:

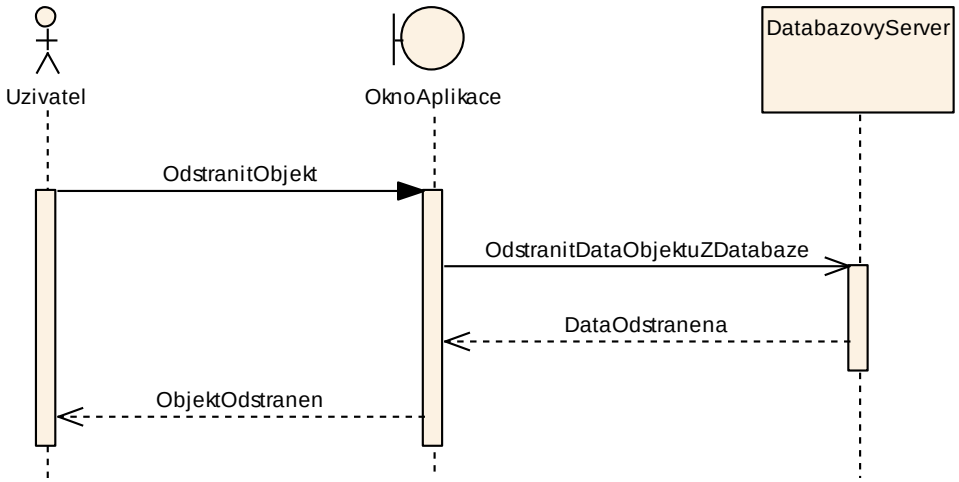
- **Přidat object**



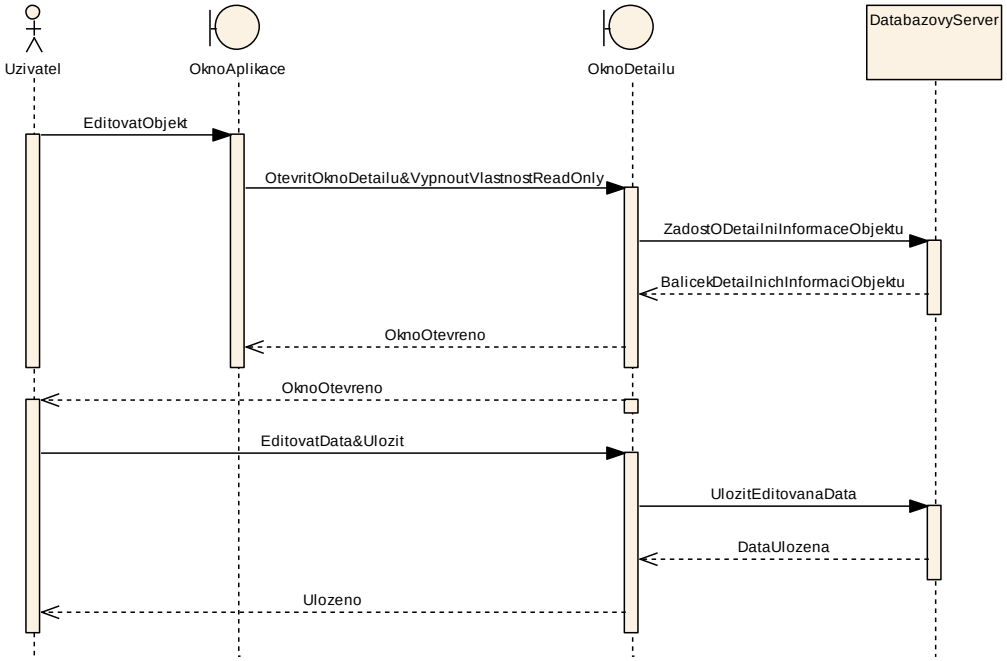
- **Odstranit object**



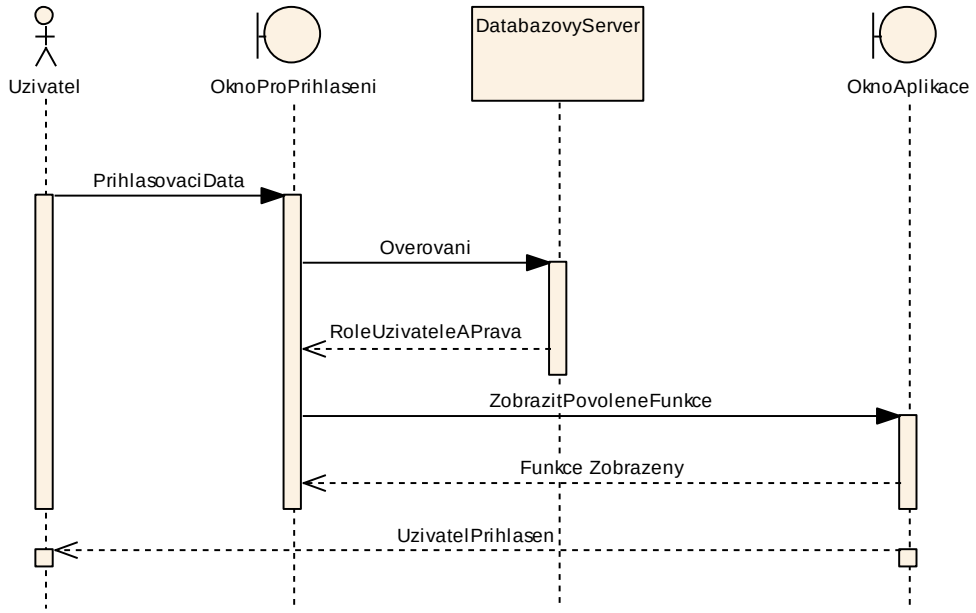
- **Odstranit objekt pomocí atributu**



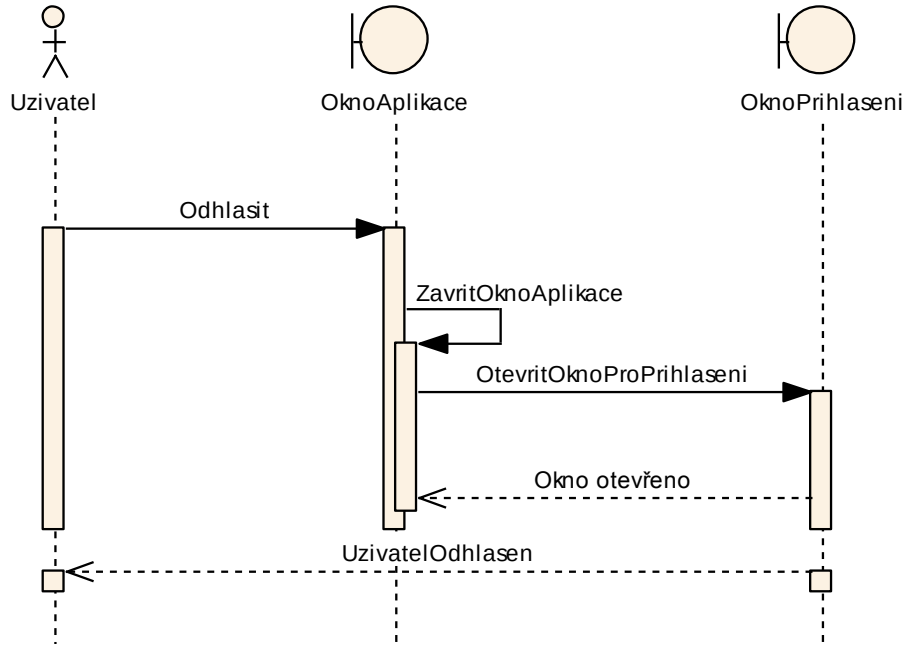
- **Editovat objekt**



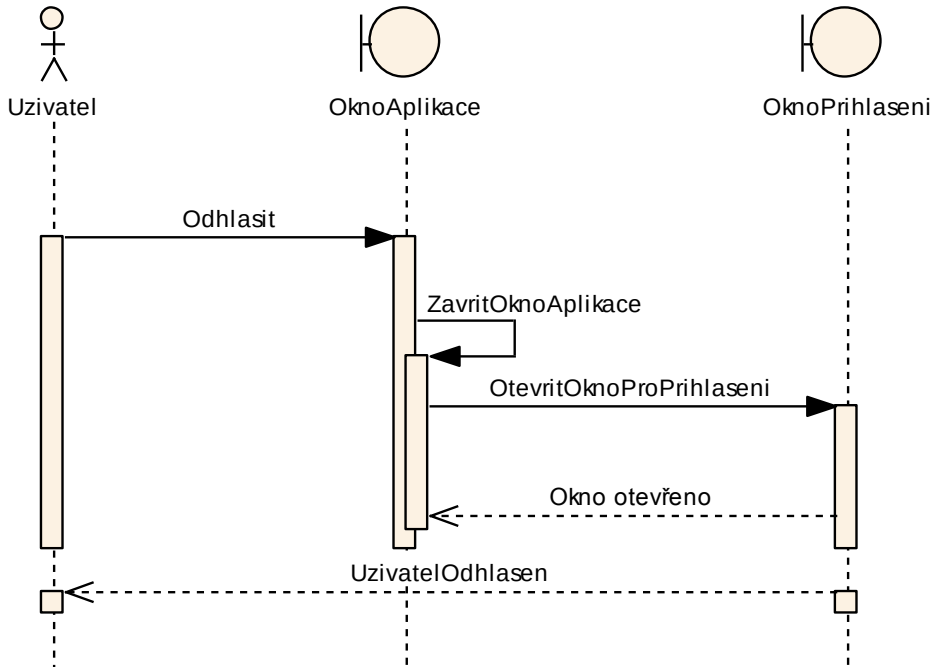
- **Přihlásit uživatele**



- **Odhlásit uživatele**



- **Zobrazit list objektu**



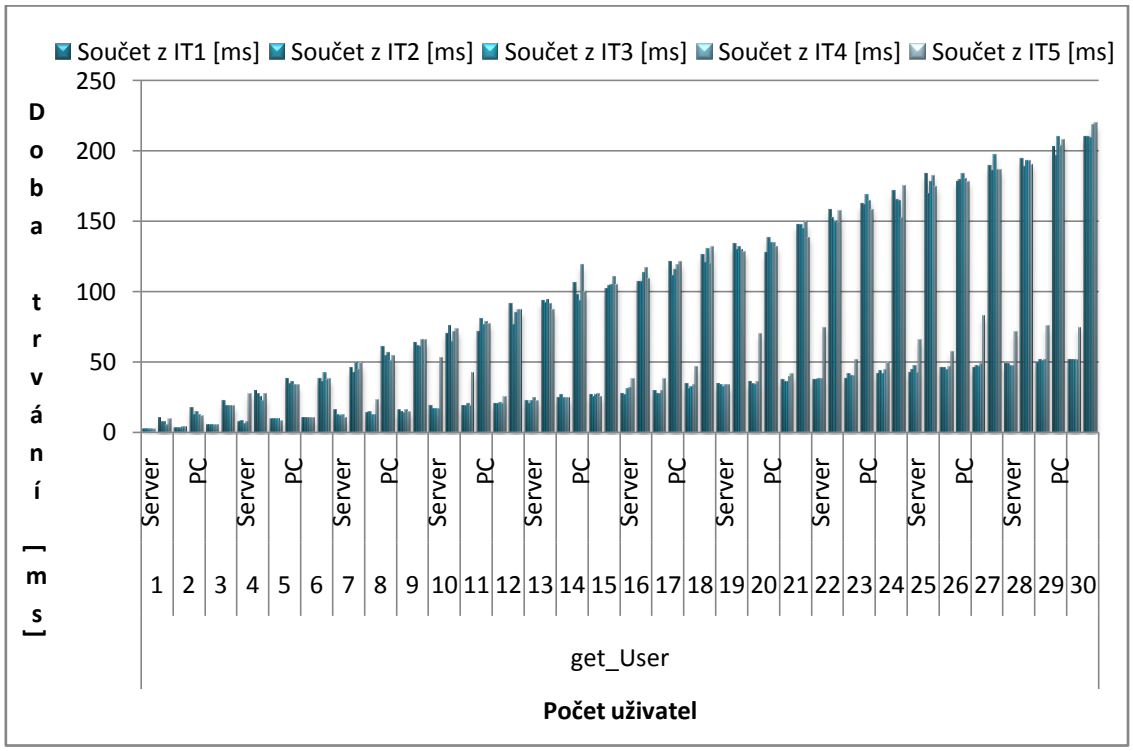
Příloha IV Výsledky zátěžového testování přístupu k datům

- **1 – 30 uživatel, postupně, krok = 1**

Tabulka naměřených hodnot:

		2x4GHz + 6GB DDR2 RAM (Server)					AMD 850MHz + 512MB SD RAM (PC)				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
1	get_User	2	2	2	2	2	10	7	7	5	9
2	get_User	3	3	3	4	4	17	12	14	12	11
3	get_User	5	5	5	5	5	22	18	18	18	19
4	get_User	7	8	6	7	27	29	27	25	22	27
5	get_User	9	9	9	9	8	38	34	36	33	33
6	get_User	10	10	10	10	10	38	36	42	37	38
7	get_User	15	12	11	12	10	45	42	49	44	49
8	get_User	13	14	12	12	23	60	54	56	50	54
9	get_User	15	14	13	15	14	63	61	60	65	65
10	get_User	19	16	16	16	52	70	75	64	71	73
11	get_User	18	18	20	19	42	71	80	76	78	77
12	get_User	20	20	21	20	25	91	76	84	86	86
13	get_User	22	20	22	24	22	93	92	94	91	86
14	get_User	24	26	24	24	24	106	97	93	118	99
15	get_User	26	25	26	27	25	101	103	105	110	105
16	get_User	27	26	30	31	38	107	107	113	116	109
17	get_User	29	27	27	29	38	121	111	115	118	121
18	get_User	34	30	32	33	46	126	120	130	119	131
19	get_User	34	33	32	33	33	133	129	131	129	128
20	get_User	36	34	33	35	70	127	138	134	134	131
21	get_User	37	36	36	39	41	147	147	144	149	138
22	get_User	37	37	38	38	74	158	152	149	148	157
23	get_User	38	41	40	40	51	162	161	168	164	158
24	get_User	41	43	41	44	49	171	165	164	152	175
25	get_User	42	44	47	42	65	183	169	178	182	174
26	get_User	45	45	44	46	57	178	179	183	180	178
27	get_User	45	47	46	48	82	189	185	197	186	186
28	get_User	48	48	47	47	71	194	188	193	193	190
29	get_User	49	51	50	51	75	202	196	210	203	208
30	get_User	51	51	51	51	74	210	210	209	218	219

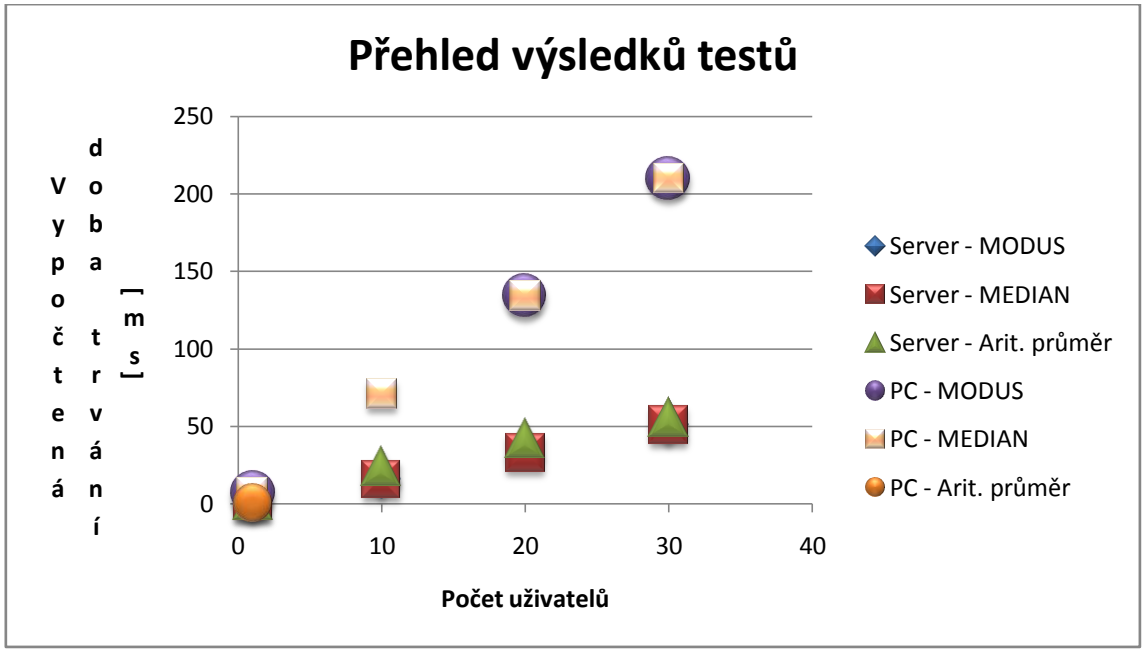
Kontingenční graf naměřených hodnot:



Tabulka vypočtených hodnot:

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
1	2	2	2,00	7	7	7,6
10	16	16	23,8	#N/A	71	70,6
20	#N/A	33	41,6	134	134	132,8
30	51	51	55,6	210	210	213,2

Graf vypočtených hodnot:

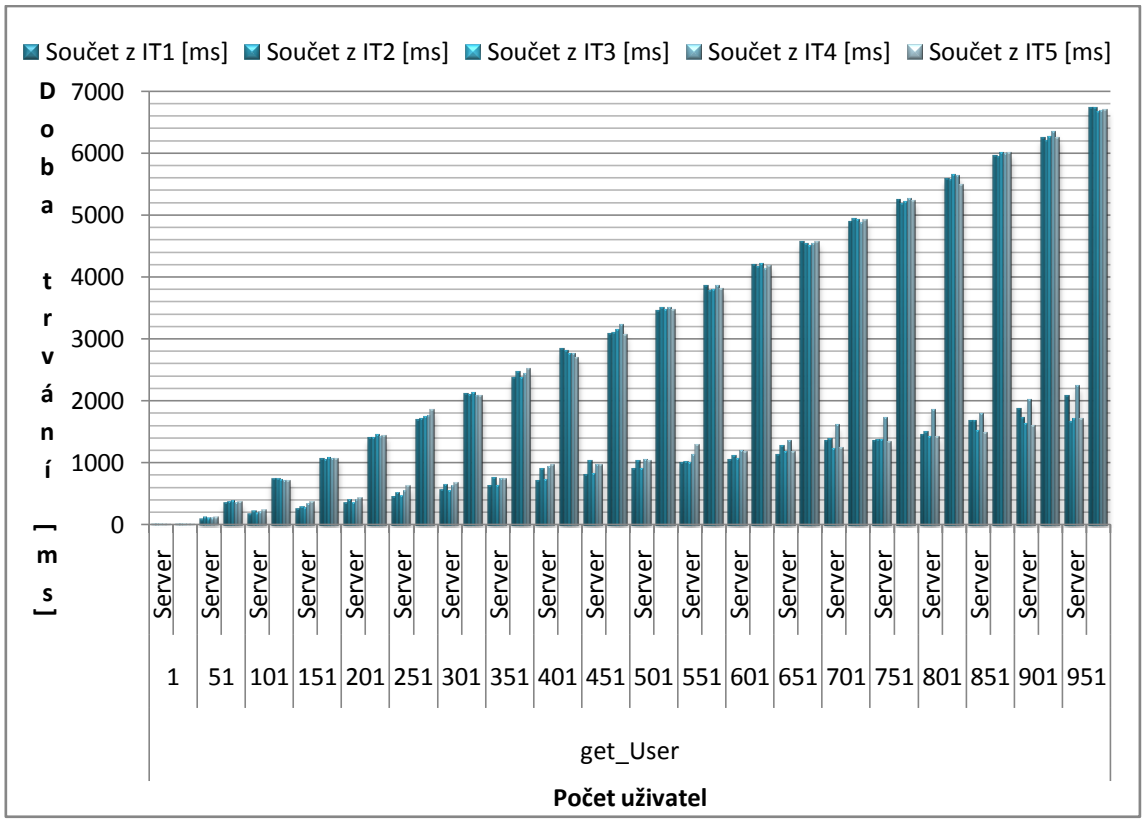


- **1 – 1000 uživatel, postupně, krok = 50**

Tabulka naměřených hodnot:

		2x4GHz + 6GB DDR2 RAM (Server)					AMD 850MHz + 512MB SD RAM (PC)				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
1	get_User	9	2	2	4	2	7	7	8	7	9
51	get_User	89	109	99	105	112	349	363	376	353	365
101	get_User	169	203	177	194	224	730	733	714	710	708
151	get_User	258	288	275	328	371	1059	1054	1071	1070	1058
201	get_User	350	396	354	399	428	1390	1401	1446	1420	1435
251	get_User	450	502	458	532	621	1681	1696	1736	1741	1854
301	get_User	544	637	542	608	673	2098	2085	2124	2073	2070
351	get_User	617	751	635	741	736	2372	2454	2370	2434	2512
401	get_User	713	890	723	922	949	2837	2804	2757	2749	2685
451	get_User	804	1029	814	957	952	3083	3091	3146	3214	3069
501	get_User	890	1028	891	1045	1026	3459	3497	3469	3501	3471
551	get_User	983	995	989	1118	1275	3854	3782	3792	3863	3808
601	get_User	1050	1110	1059	1186	1178	4198	4165	4215	4132	4183
651	get_User	1126	1260	1178	1342	1169	4563	4532	4484	4537	4563
701	get_User	1342	1374	1214	1603	1234	4887	4944	4923	4865	4919
751	get_User	1339	1350	1355	1722	1332	5241	5173	5212	5253	5224
801	get_User	1453	1490	1417	1855	1421	5579	5567	5641	5625	5482
851	get_User	1671	1667	1508	1798	1483	5945	5938	5993	5987	5989
901	get_User	1870	1717	1626	2012	1596	6238	6201	6255	6343	6241
951	get_User	2075	1653	1694	2236	1698	6729	6729	6659	6678	6702

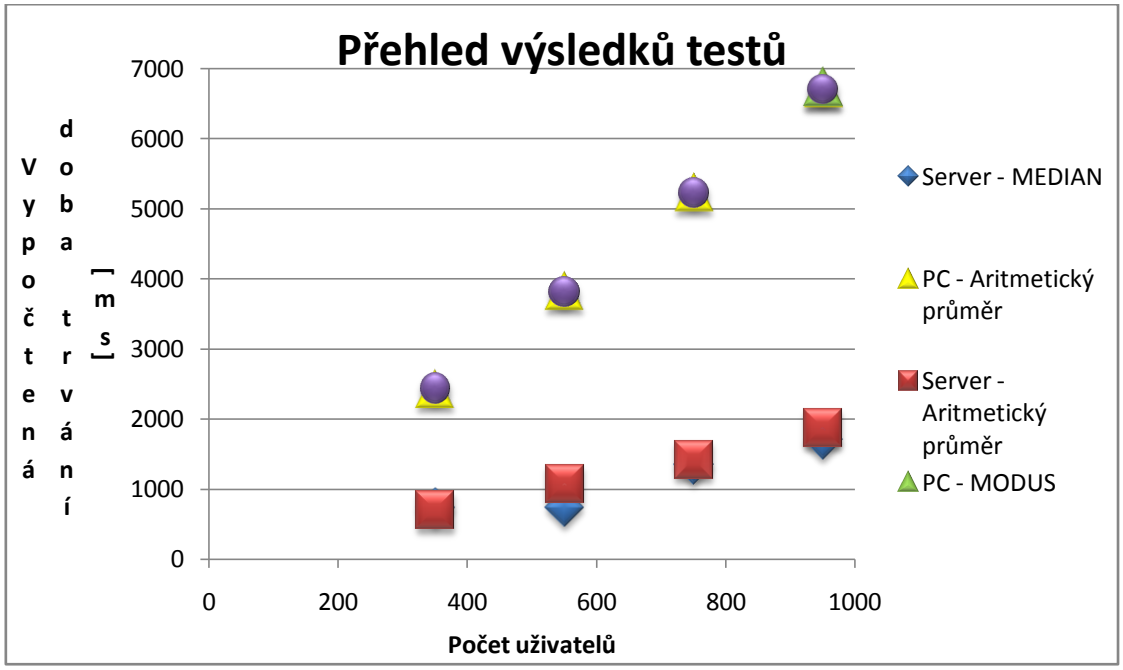
Kontingenční graf naměřených hodnot:



Tabulka vypočtených hodnot:

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
351	#N/A	736	696	#N/A	2434	2428,4
551	#N/A	736	1072	#N/A	3808	3819,8
751	#N/A	1350	1419,6	#N/A	5224	5220,6
951	#N/A	1698	1871,2	6729	6702	6699,4

Graf vypočtených hodnot:

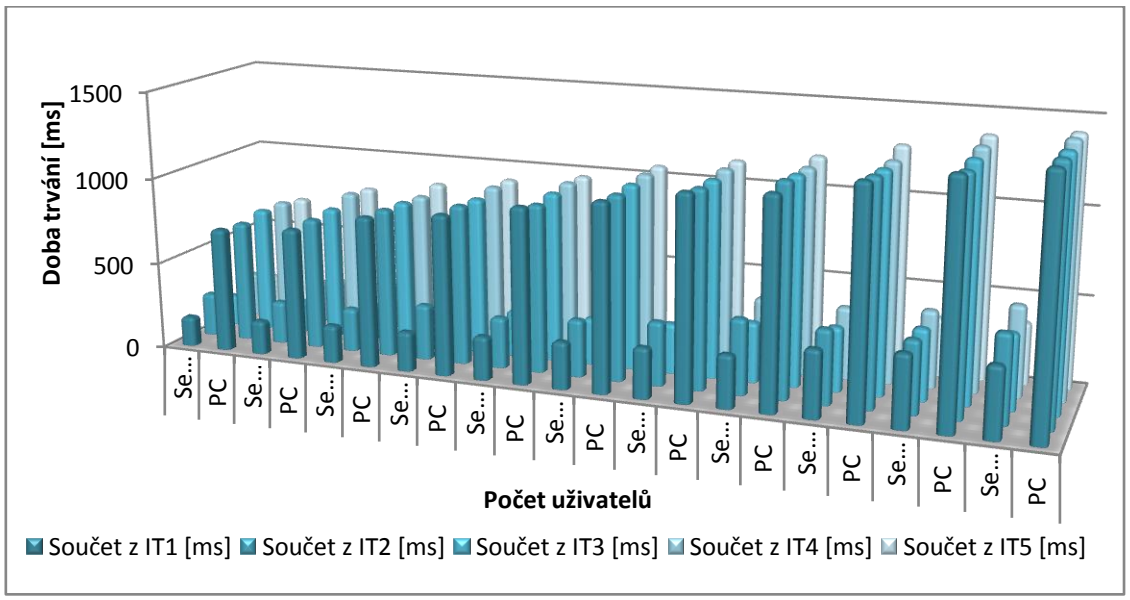


- **100 – 200 uživatel, postupně, krok = 10**

Tabulka naměřených hodnot:

		2x4GHz + 6GB DDR2 RAM (Server)					AMD 850MHz + 512MB SD RAM (PC)				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
100	get_User	173	249	178	254	180	715	697	724	720	690
110	get_User	194	245	190	249	192	755	759	769	812	792
120	get_User	215	250	213	277	210	865	856	845	830	855
130	get_User	227	318	230	319	226	917	917	903	923	916
140	get_User	250	292	267	322	252	1000	960	973	984	972
150	get_User	270	328	269	341	270	1068	1053	1059	1068	1067
160	get_User	289	365	288	347	285	1154	1125	1121	1132	1132
170	get_User	301	431	344	415	312	1184	1203	1184	1174	1192
180	get_User	383	423	366	411	325	1284	1261	1248	1245	1287
190	get_User	412	415	407	441	333	1355	1321	1336	1356	1373
200	get_User	401	502	432	518	356	1418	1416	1411	1428	1417

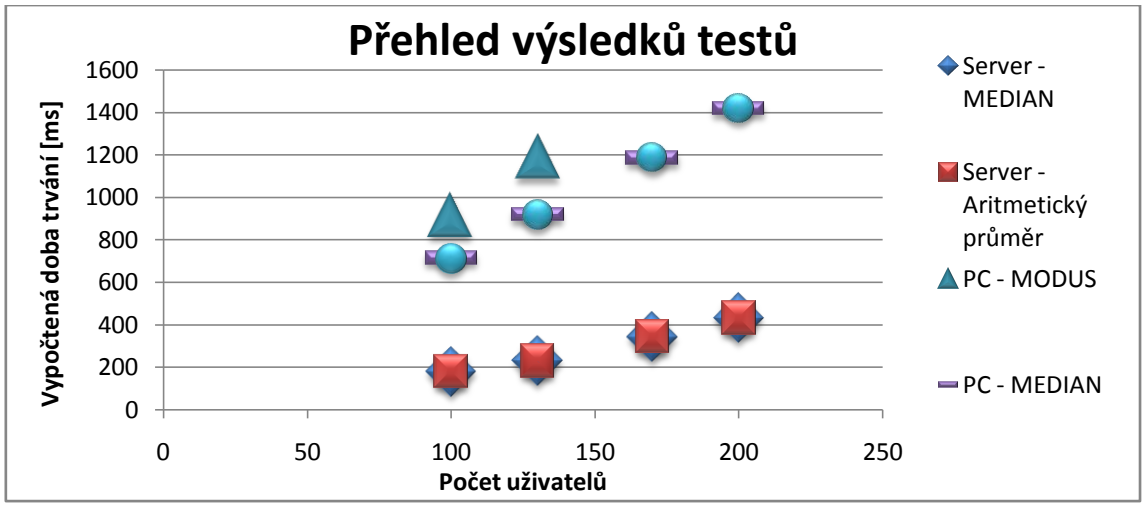
Kontingenční graf naměřených hodnot:



Tabulka vypočtených hodnot:

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
100	#N/A	180	206,8	#N/A	715	709,2
130	#N/A	230	264	917	917	915,2
170	#N/A	344	360,6	1184	1184	1187,4
200	#N/A	432	441,8	#N/A	1417	1418

Graf vypočtených hodnot:

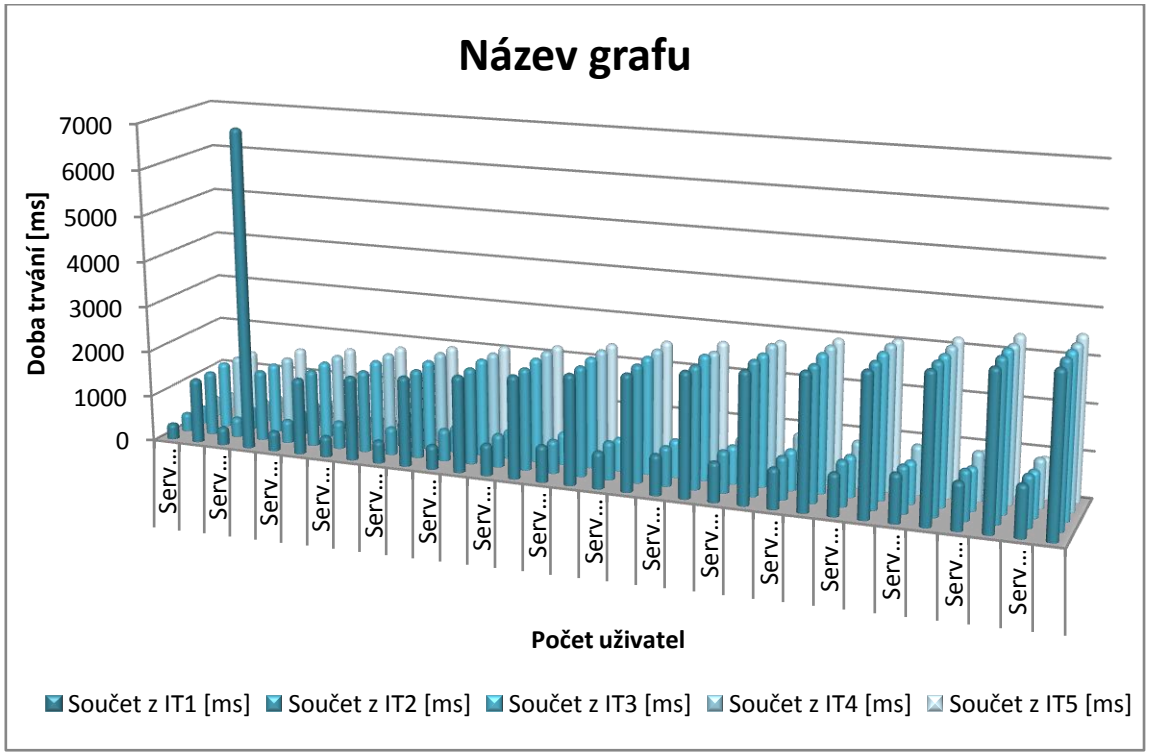


- **200 – 500 uživatel, postupně, krok = 20**

Tabulka naměřených hodnot:

		2x4GHz + 6GB DDR2 RAM (Server)					AMD 850MHz + 512MB SD RAM (PC)				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
200	get_User	350	411	388	433	353	1413	1406	1449	1416	1381
220	get_User	391	430	487	473	394	7268	1548	1535	1517	1563
240	get_User	432	502	539	554	442	1672	1684	1700	1686	1681
260	get_User	457	603	537	547	461	1837	1793	1852	1845	1836
280	get_User	491	603	570	580	505	1965	1939	1972	1987	1962
300	get_User	531	698	590	643	534	2108	2097	2125	2099	2087
320	get_User	695	718	633	666	578	2236	2219	2264	2261	2215
340	get_User	798	725	706	669	613	2391	2384	2407	2403	2376
360	get_User	802	830	716	776	653	2522	2542	2543	2536	2549
380	get_User	880	860	824	784	676	2697	2645	2706	2607	2660
400	get_User	863	907	826	825	727	2835	2880	2851	2890	2794
420	get_User	876	923	879	1037	759	2946	2913	2979	2975	2961
440	get_User	915	983	896	1009	782	3089	3097	3120	3128	3066
460	get_User	1054	1021	919	1063	821	3220	3226	3195	3206	3213
480	get_User	1023	1076	932	1063	870	3391	3435	3428	3360	3402
500	get_User	1117	1113	1005	1061	891	3489	3528	3516	3492	3518

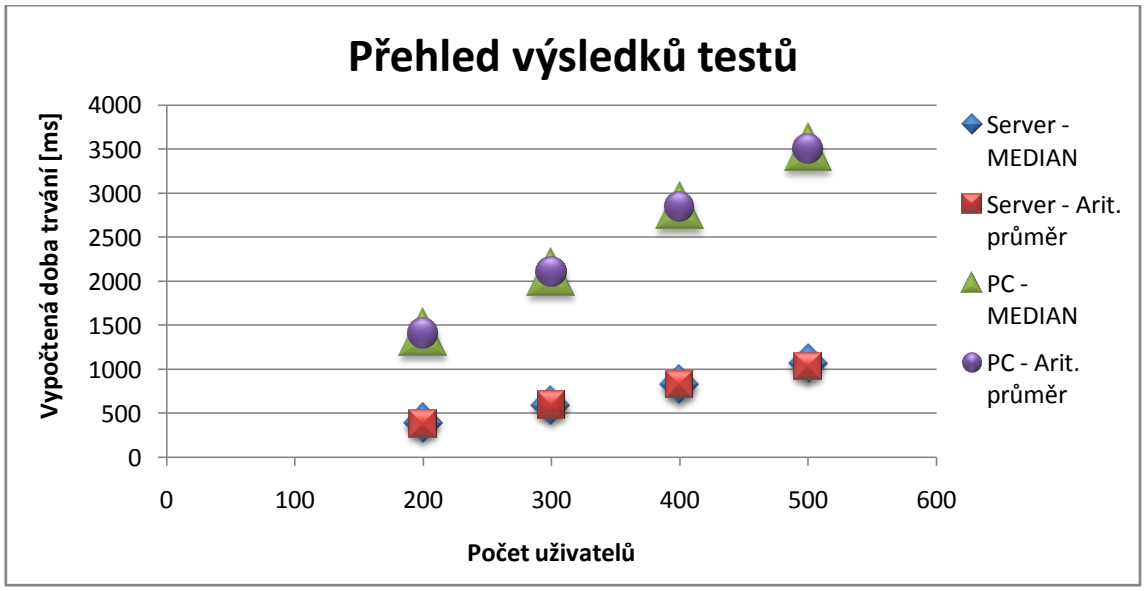
Kontingenční graf naměřených hodnot:



Tabulka vypočtených hodnot:

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
200	#N/A	388	387	#N/A	1413	1413
300	#N/A	590	599,2	#N/A	2099	2103,2
400	#N/A	826	829,6	#N/A	2851	2850
500	#N/A	1061	1037,4	#N/A	3516	3508,6

Graf vypočtených hodnot:

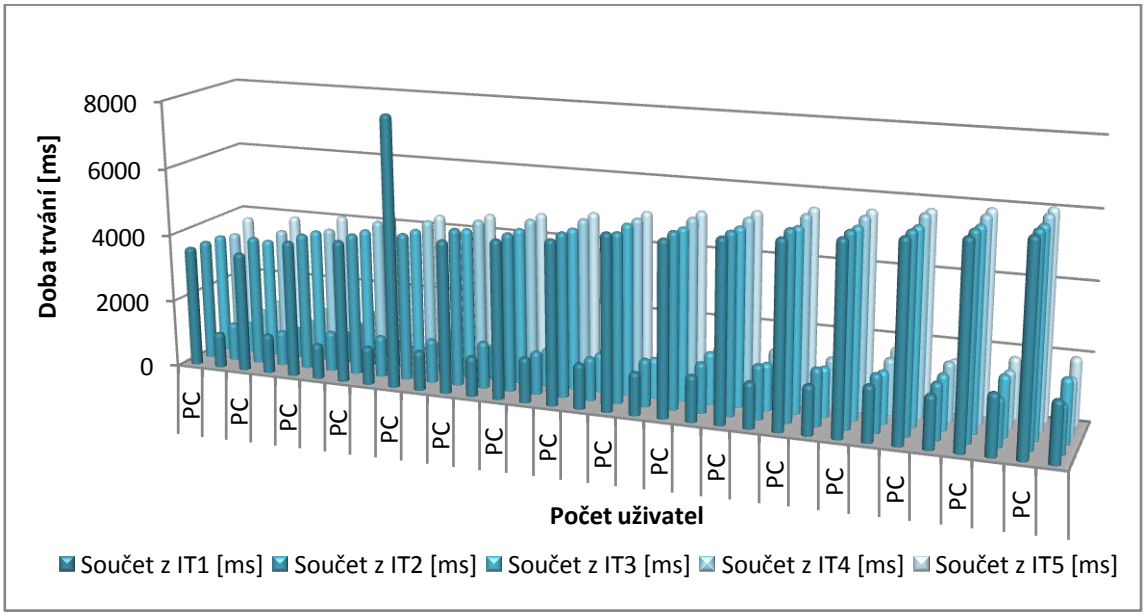


- **500 – 800 uživatel, postupně, krok = 20**

Tabulka naměřených hodnot:

		2x4GHz + 6GB DDR2 RAM (Server)					AMD 850MHz + 512MB SD RAM (PC)				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
500	get_User	1054	1128	911	1107	1163	3602	3596	3578	3456	3778,026
520	get_User	1164	1046	932	953	1115	3564	3850	3576	3689	3932,519
540	get_User	1044	1188	980	1055	1214	4061	4083	3993	3865	4077,318
560	get_User	1131	1220	1002	1077	1237	4216	4227	4161	4237	4238,153
580	get_User	1177	1291	1038	1202	1280	8128	4374	4311	4389	4374,01
600	get_User	1189	1384	1088	1233	1329	4516	4660	4464	4538	4533,214
620	get_User	1335	1265	1115	1351	1386	4677	4669	4606	4695	4690,297
640	get_User	1334	1271	1154	1509	1465	4821	4831	4754	4842	4846,701
660	get_User	1261	1398	1182	1454	1509	5153	4970	5026	4996	5009,161
680	get_User	1355	1473	1526	1454	1451	5119	5141	5058	5151	5148,696
700	get_User	1349	1592	1390	1548	1417	5312	5284	5220	5288	5296,319
720	get_User	1450	1677	1458	1482	1552	5424	5488	5362	5515	5520,318
740	get_User	1632	1697	1528	1637	1703	5565	5579	5509	5601	5597,073
760	get_User	1555	1634	1622	1711	1533	5732	5729	5653	5802	5746,736
780	get_User	1791	1401	1775	1645	1778	5871	5881	5805	5899	5893,291
800	get_User	1783	1425	1872	1626	1949	6042	6032	5957	6048	6042,658

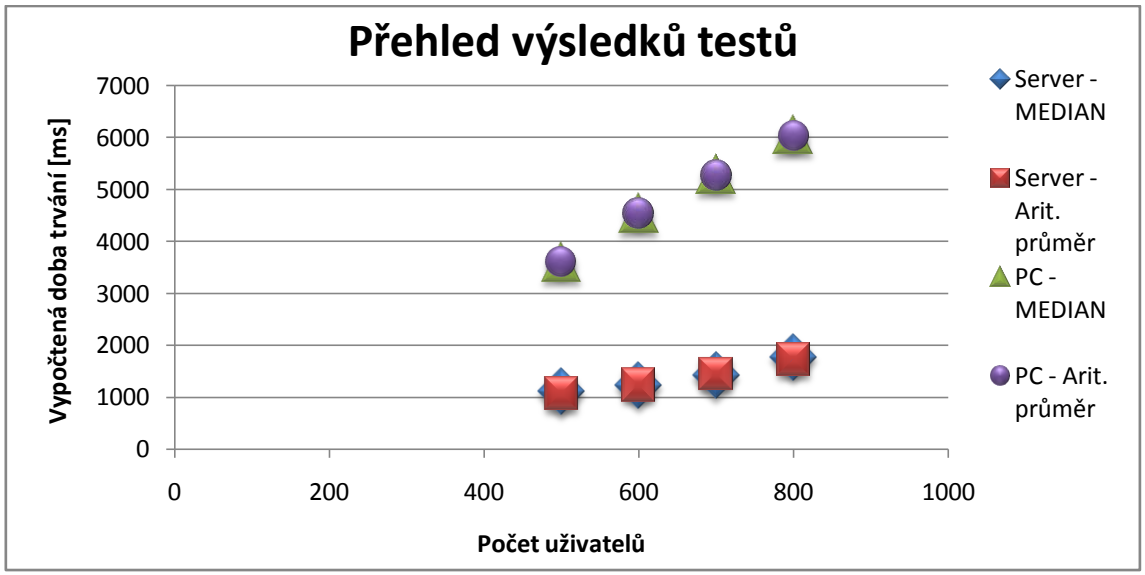
Kontingenční graf naměřených hodnot:



Tabulka vypočtených hodnot:

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
500	#N/A	1107	1072,6	#N/A	3596,021	3601,915327
600	#N/A	1233	1244,6	#N/A	4533,214	4542,270987
700	#N/A	1417	1459,2	#N/A	5287,592	5279,962062
800	#N/A	1783	1731	#N/A	6041,503	6024,197992

Graf vypočtených hodnot:

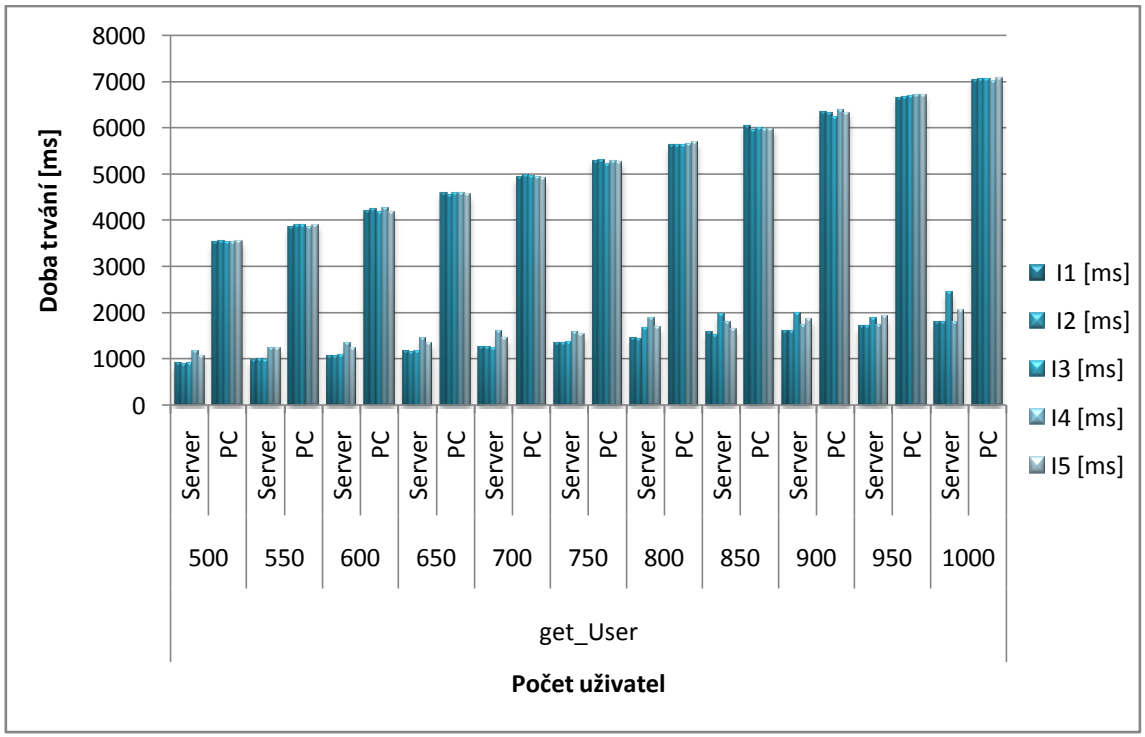


- **500 – 1000 uživatel, postupně, krok = 50**

Tabulka naměřených hodnot:

		2x4GHz + 6GB DDR2 RAM (Server)					AMD 850MHz + 512MB SD RAM (PC)				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
500	get_User	892	888	895	1176	1078	3511	3570	3526	3526	3535
550	get_User	979	1006	994	1239	1239	3861	3898	3886	3852	3887
600	get_User	1071	1076	1083	1348	1243	4212	4255	4184	4270	4192
650	get_User	1168	1154	1182	1438	1348	4574	4543	4574	4588	4548
700	get_User	1254	1265	1240	1617	1443	4927	4967	4942	4933	4911
750	get_User	1342	1346	1354	1591	1549	5289	5309	5231	5281	5272
800	get_User	1441	1425	1662	1873	1682	5616	5645	5635	5657	5696
850	get_User	1595	1526	1964	1804	1656	6023	5939	5994	5974	5978
900	get_User	1613	1604	1986	1740	1854	6344	6316	6238	6387	6324
950	get_User	1703	1705	1874	1725	1914	6643	6677	6689	6707	6713
1000	get_User	1797	1786	2434	1796	2072	7037	7058	7050	7010	7079

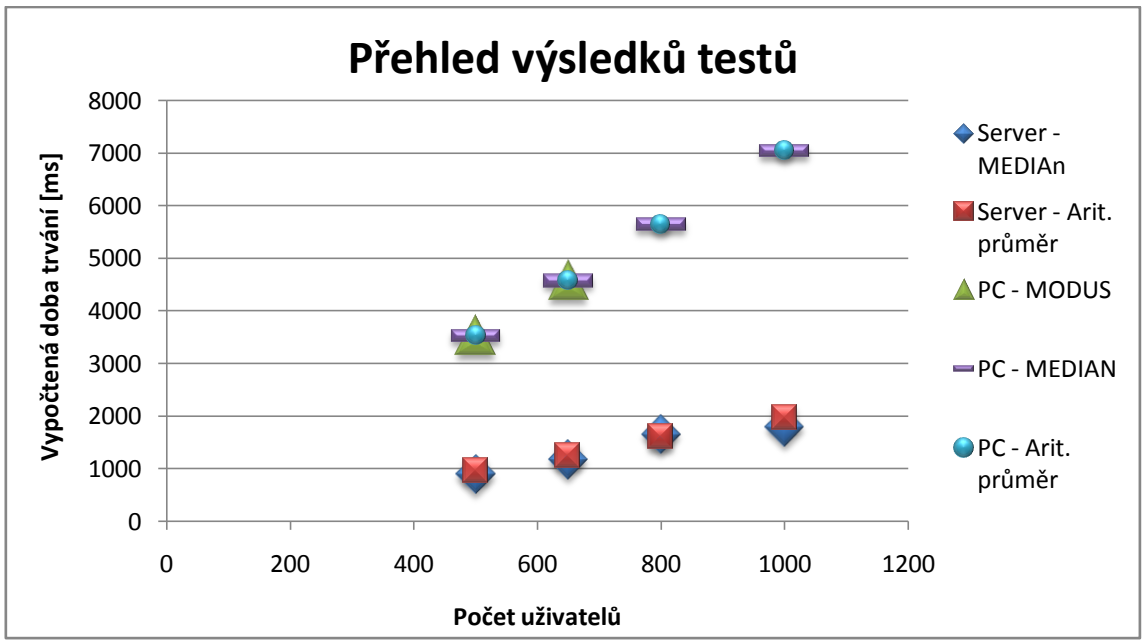
Kontingenční graf naměřených hodnot:



Tabulka vypočtených hodnot:

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
500	#N/A	895	985,8	3526	3526	3533,6
650	#N/A	1182	1258	4574	4574	4565,4
800	#N/A	1662	1616,6	#N/A	5645	5649,8
1000	#N/A	1797	1977	#N/A	7050	7046,8

Graf vypočtených hodnot:

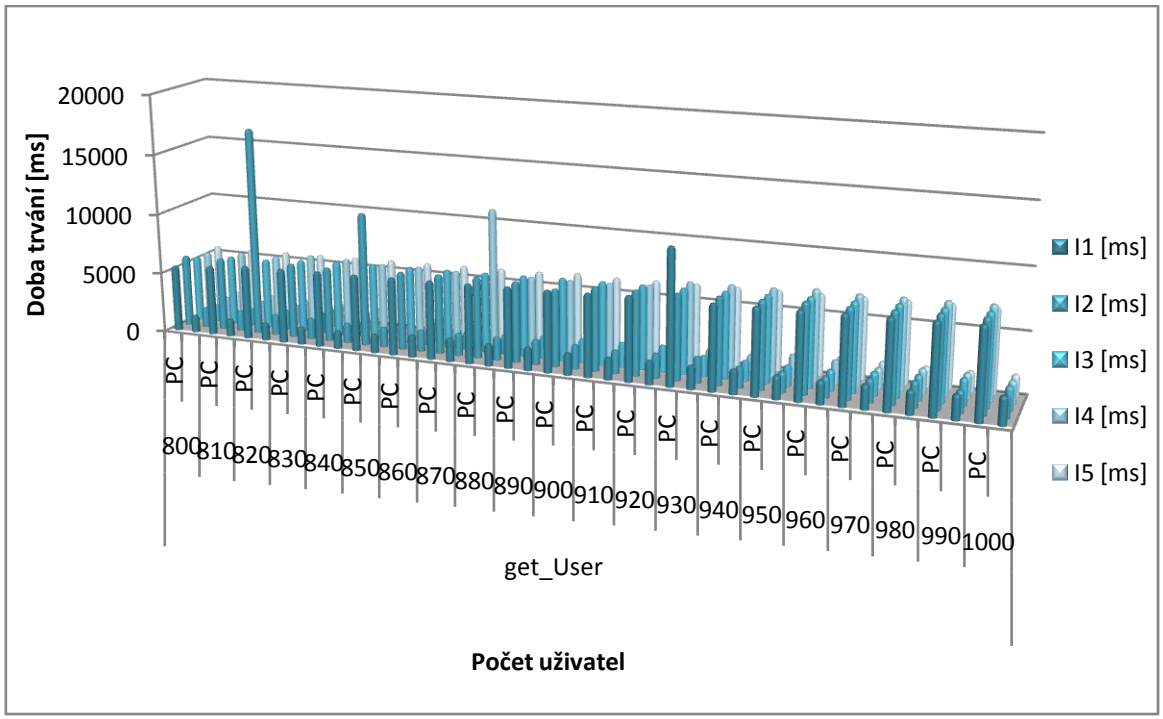


- **800 – 1000 uživatel, postupně, krok = 10**

Tabulka naměřených hodnot

		2x4GHz + 6GB DDR2 RAM					AMD 850MHz + 512MB SD RAM				
pocet uzivatelu	Volaná metoda	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]	IT1 [ms]	IT2 [ms]	IT3 [ms]	IT4 [ms]	IT5 [ms]
800	get_User	1436	1661	1425	1666	1652	5552	5988	5474	5458	5595
810	get_User	1470	1711	1437	1627	1719	5821	6060	5815	5745	5624
820	get_User	1482	1731	1672	1658	1636	6129	17254	5830	5718	5643
830	get_User	1481	1752	1850	1766	1729	6195	6214	6098	6095	5719
840	get_User	1511	1686	1488	1722	1744	6301	6200	6294	6077	5784
850	get_User	1561	1681	1521	1716	1725	6337	11033	6369	5955	5843
860	get_User	1869	1773	1537	1641	1806	6428	6473	6439	6026	5922
870	get_User	1957	1876	1537	1625	1732	6500	6560	6521	5978	5983
880	get_User	1867	1929	1611	1782	1774	6573	6702	6581	11437	6075
890	get_User	1934	2066	1866	1668	1887	6650	6637	6672	6118	6129
900	get_User	1860	2014	2001	1866	1527	6714	6362	6799	6179	6324
910	get_User	1858	1944	2053	1739	1576	6801	6866	6829	6251	6265
920	get_User	1972	2052	2185	1718	1592	6949	6925	6896	6430	6349
930	get_User	1951	2144	1718	1763	1604	11119	6996	6957	7019	6403
940	get_User	2035	1738	1693	1662	1629	7020	7096	7039	7108	6473
950	get_User	1930	1712	1698	2058	1665	7098	7146	7108	7194	6544
960	get_User	1935	1715	1711	1965	1697	7177	7359	7266	7426	6604
970	get_User	2008	1929	1922	1716	1702	7246	7294	7268	7348	6663
980	get_User	1977	2136	2075	2049	1684	7327	7373	7393	7422	6749
990	get_User	2069	1805	2211	2074	1699	7400	7454	7418	7488	6822
1000	get_User	2334	1789	1984	1866	1803	7480	7550	7482	7566	6891

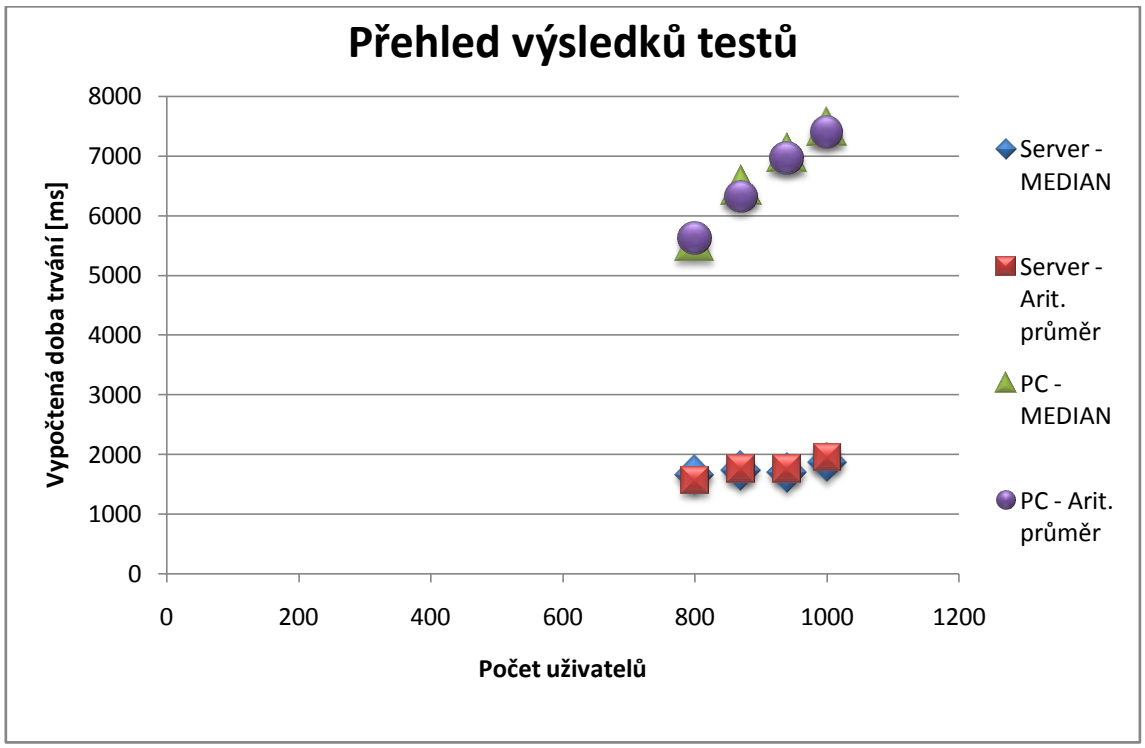
Kontingenční graf naměřených hodnot:



Tabulka vypočtených hodnot:

Vypočtené hodnoty:	2x4GHz + 6GB DDR2 RAM (Server)			AMD 850MHz + 512MB SD RAM (PC)		
Počet uživatelů	modus	medián	aritmetický průměr	modus	medián	aritmetický průměr
800	#N/A	1652	1568	#N/A	5552,122	5613,606869
870	#N/A	1732	1745,4	#N/A	6500	6308,383384
940	#N/A	1693	1751,4	#N/A	7039	6947,229606
1000	#N/A	1866	1955,2	#N/A	7482	7393,571418

Graf vypočtených hodnot:



- **1000 – 1000 uživatel, random**

Tabulka naměřených hodnot pro dobu testování 10 min

doba testovani: 10 min		2x4GHz + 6GB DDR2 RAM				
		IT1	IT2	IT3	IT4	IT5
volana metoda	poc. Uziv.	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]
get_User	1000	1901	1966	1960	1908	1946
get_User	1000	1924	1933	1933	1922	2005
get_User	1000	1901	1952	1952	1879	2009
get_User	1000	2048	1956	1956	1914	2027
get_User	1000	1949	1971	1971	1902	2015
get_User	1000	1914	1918	1918	1972	2987
get_User	1000	1876	1937	1937	1903	2011
get_User	1000	1950			1914	2000
get_User	1000					1992
get_User	1000					1982

AMD 850MHz + 512MB SD RAM				
IT1	IT2	IT3	IT4	IT5
doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]	doba trvani [ms]
7069	7106	7078	7103	7021
6039	7068	7049	7001	6981
6976	7005	6976	6987	6979
6980	7061	7050	6952	7004
7000	7011	7024	7033	7060
7046	7030	7084	7130	6983
7004	7054	7020		7037
7009	7071			7019

Výstupní grafy z testovací aplikace – 2x4GHz + 6GB DDR2 RAM:

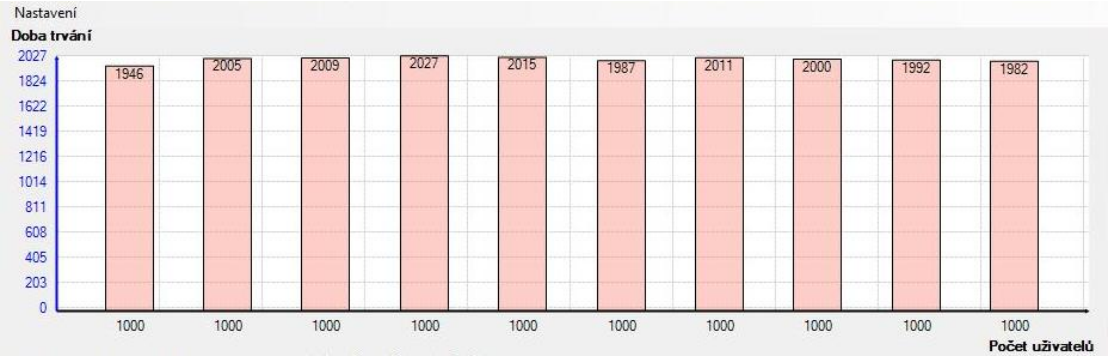
Iterace č. 1

Iterace č. 2

Iterace č. 3

Iterace č. 4

Iterace č. 5

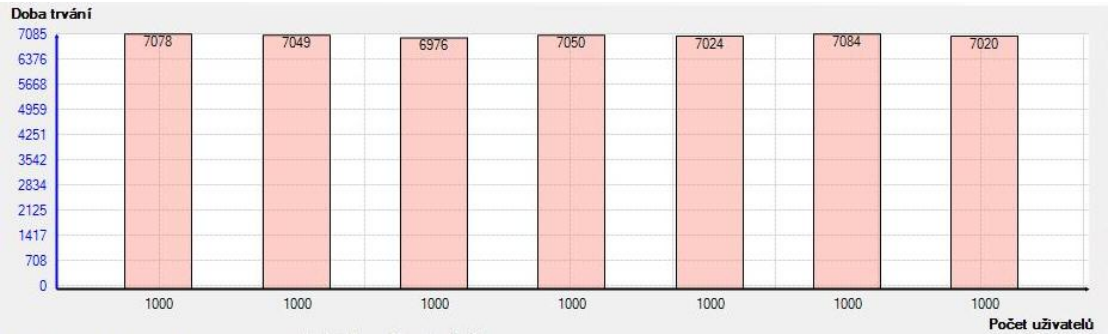


Výstupní grafy z testovací aplikace – AMD 850MHz + 512MB SD RAM:

Iterace č. 1

Iterace č. 2

Iterace č. 3



Iterace č. 4

Iterace č. 5

- **1 – 1000 uživatel, random**

Tabulka naměřených hodnot pro dobu testování 3 min:

doba testovani:	2x4GHz + 6GB DDR2 RAM									
3 min	IT1		IT2		IT3		IT4		IT5	
volana metoda	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]
get_User	496	950	5	20	187	346	597	1173	128	268
get_User	104	194	381	757	879	1783	325	637	287	590
get_User	705	1323	516	977	458	867	303	587	953	1875
get_User	296	580	413	798	974	1907	520	1036	942	1859
get_User	673	1273	76	142			661	1292	61	140
get_User	531	1037	627	1214						
get_User			116	220						
get_User			69	146						
get_User			749	1459						

Tabulka naměřených hodnot pro dobu testování 5 min:

doba testovani:	2x4GHz + 6GB DDR2 RAM									
5 min	IT1		IT2		IT3		IT4		IT5	
volana metoda	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]
get_User	189	346	108	209	826	1586	245	456	217	405
get_User	408	787	349	658	658	1262	123	229	342	680
get_User	815	1524	565	1087	39	73	469	905	673	1375
get_User	838	1587	455	874	68	128	66	146	458	918
get_User	688	1327	473	900	946	1904	868	1707	312	643
get_User	786	1491	724	1408	745	1494	882	1694	485	948
get_User	105	202	541	1013	100	196	629	1211	747	1422
get_User	333	629	518	1022	432	823	559	1093	721	1345
get_User	262	561	948	1855	318	656	833	1598	904	1710

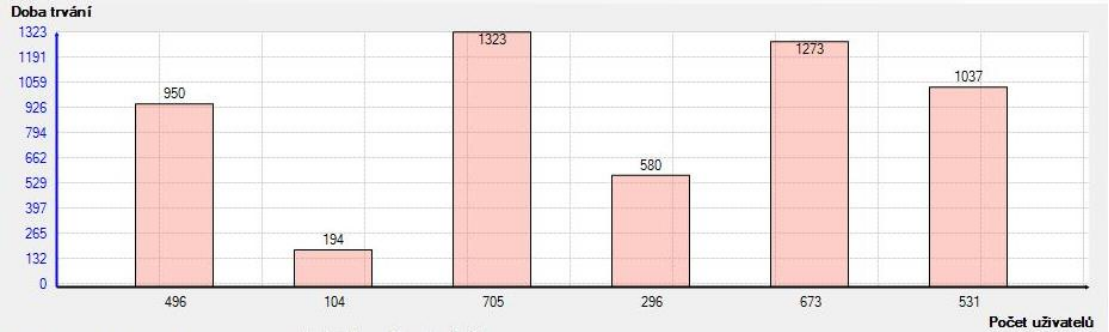
Tabulka naměřených hodnot pro dobu testování 10 min:

doba testovani:	2x4GHz + 6GB DDR2 RAM									
10 min	IT1		IT2		IT3		IT4		IT5	
volana metoda	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]
get_User	791	1512	728	1442	538	1035	82	155	372	711
get_User	228	445	53	96	807	1553	473	916	772	1497
get_User	395	832	823	1573	444	883	31	80	494	961
get_User	655	1214	732	1370	396	748	42	80	655	1332
get_User	383	718	496	953	963	1825	621	1186	934	1809
get_User	302	582	758	1551	567	1062	411	847	597	1138
get_User	303	582	888	1716	792	560	640	1225	638	1236
get_User	704	1363	554	1116	289	1717	661	1288	742	1429
get_User	315	677	311	589	906	1064	514	967	315	656
get_User	177	359	10	31	555	526	863	1669	22	53
get_User	622	1211	469	892	268	1013	275	564	962	1887
get_User	141	290	765	1487	530	1449	749	1491	62	179
get_User	73	136	435	688	762	1037	726	1428	533	1027
get_User	92	192	879	1695	537	420	48	108	621	1173
get_User	657	1268	705	1339	218		483	936	864	1631
get_User	354	719	339	646			359	697	28	51
get_User	715	1363	498	1000			988	1942		
get_User	605	1210					4	11		
get_User	760	1488					749	1475		
get_User	695	1362					298	578		

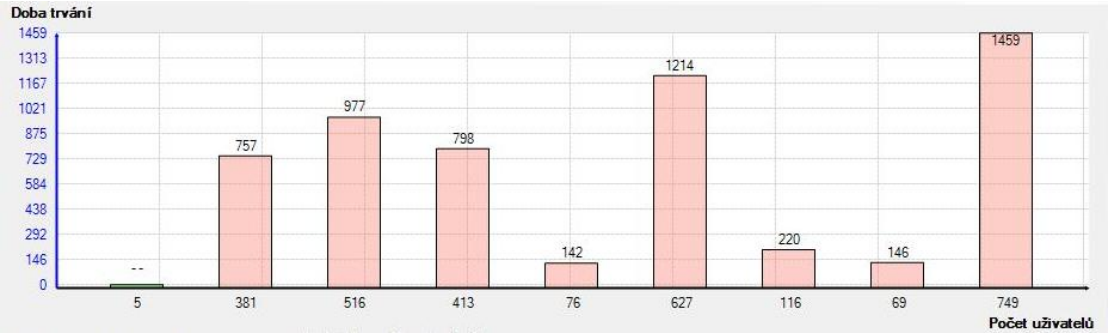
AMD 850MHz + 512MB SD RAM									
IT1		IT2		IT3		IT4		IT5	
poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]	poc. Uziv.	doba trvani [ms]
296	2088	772	5441	899	6306	502	3594	342	2383
229	1615	959	6671	86	593	960	6792	898	6316
83	591	520	3683	763	5365	142	1018	288	2029
596	4189	473	3314	474	3336	730	5151	787	5545
764	5366	277	1987	650	4592	152	1049	575	4063
645	4523	219	1536	603	4213	48	330	405	2855
211	1467	717	5077	1	9	553	3886	587	4120
461	3228	334	2338	499	3533	588	4118	209	1480
630	4389	319	2247	779	5496	652	4562	707	4998
261	1851	968	6836	874	6199	674	4699	8	46
970	6931	799	5615	810	5726	384	2726	826	5783
611	4275	780	5482	320	2305	576	4109	303	2120
190	1330	979	6882	577	4062	654	4573	494	3480
107	743			240	1683	304	2149	977	6862
899	6279					575	4041		
942	6663								

Výstupní grafy z testovací aplikace pro dobu trvání 3 min – 2x4GHz + 6GB DDR2 RAM:

Iterace č. 1

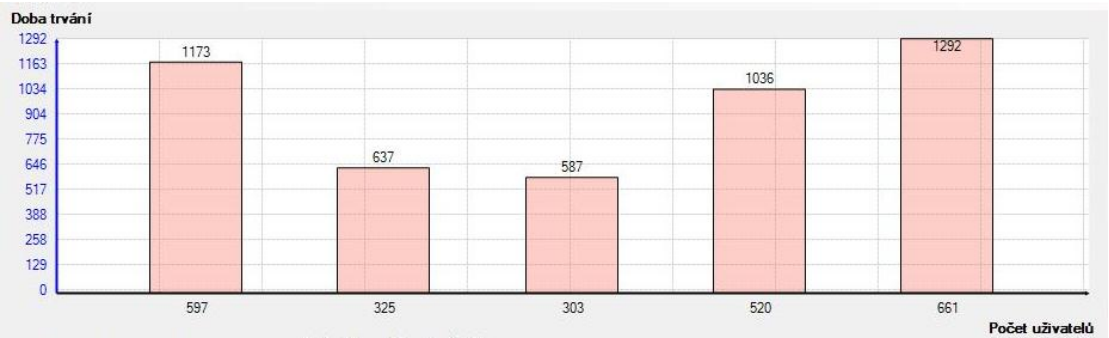


Iterace č. 2



Iterace č. 3

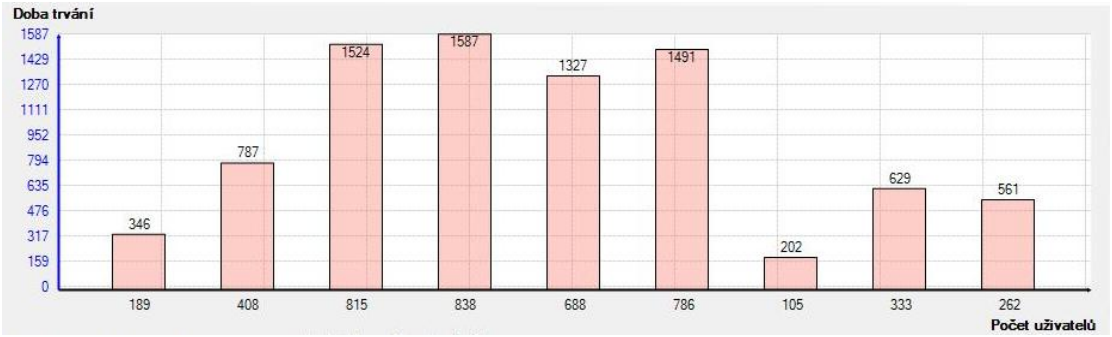
Iterace č. 4



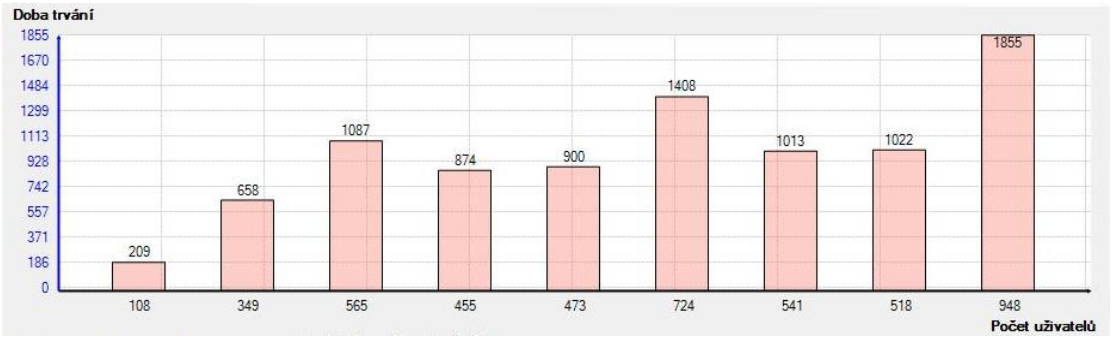
Iterace č. 5

Výstupní grafy z testovací aplikace pro dobu trvání 5 min – 2x4GHz + 6GB DDR2 RAM:

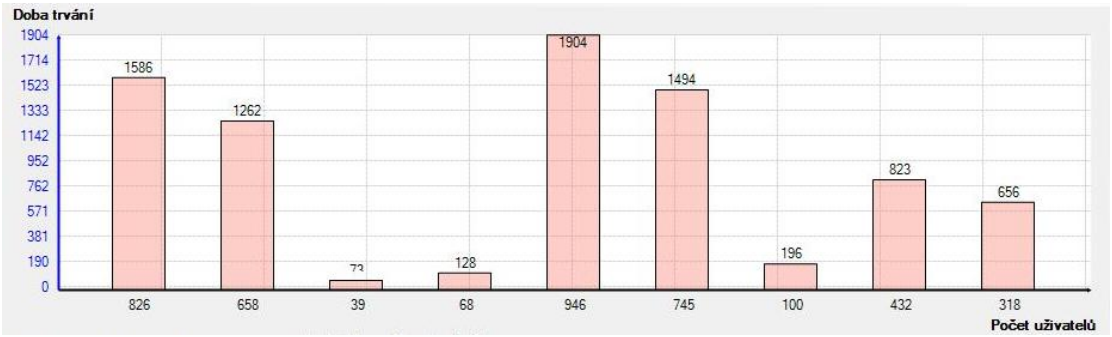
Iterace č. 1



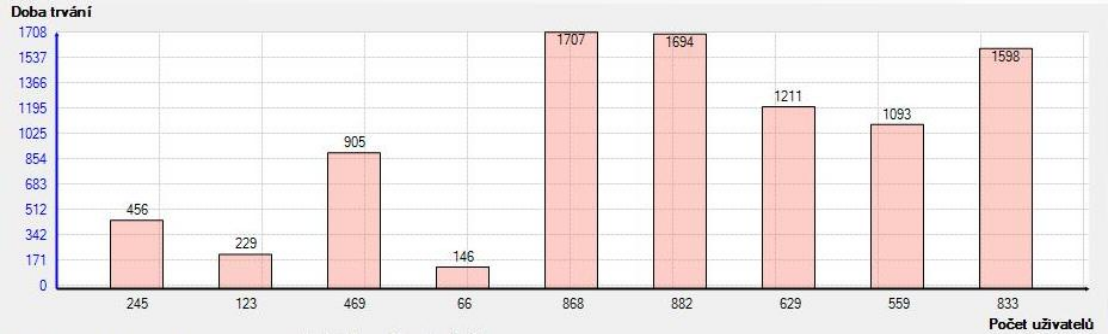
Iterace č. 2



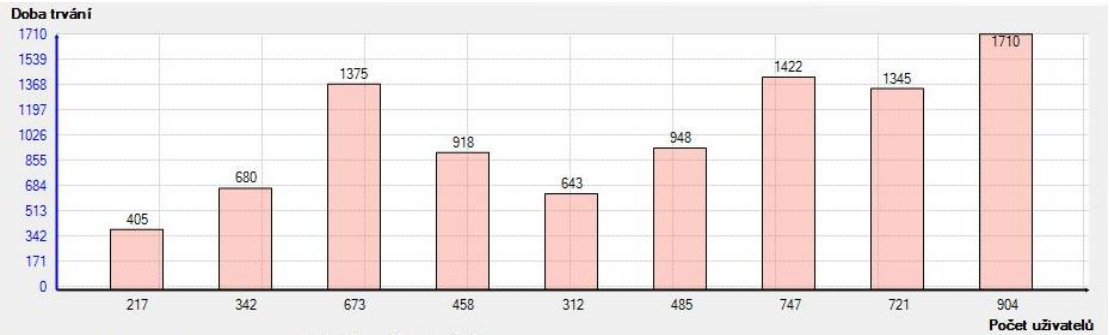
Iterace č. 3



Iterace č. 4

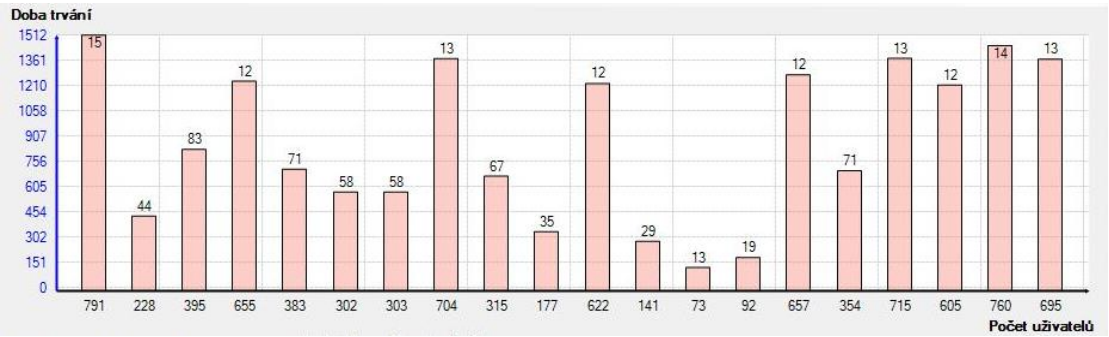


Iterace č. 5

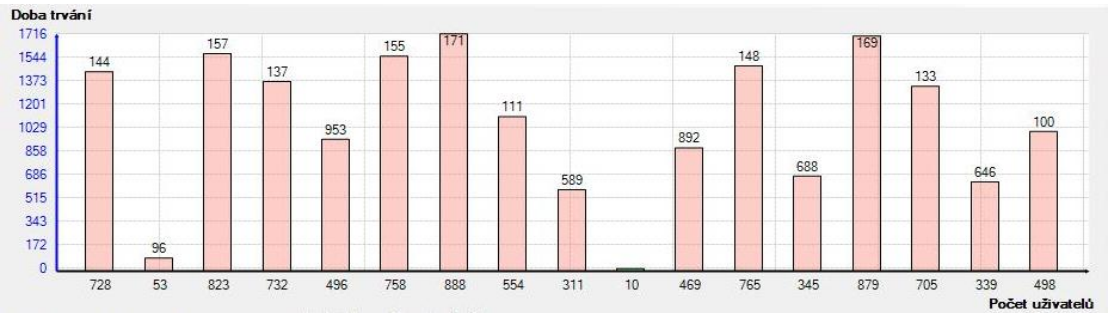


Výstupní grafy z testovací aplikace pro dobu trvání 10 min – 2x4GHz + 6GB DDR2 RAM:

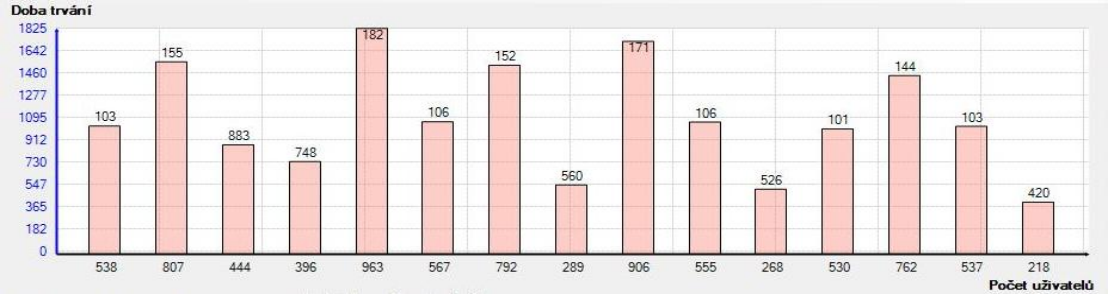
Iterace č. 1



Iterace č. 2



Iterace č. 3

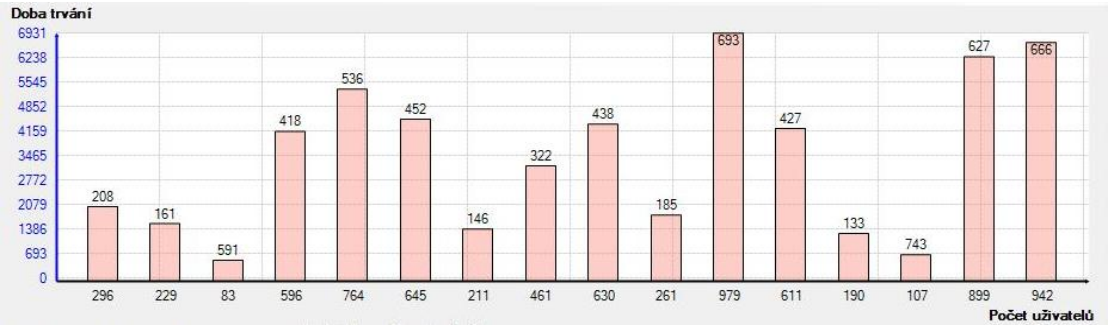


Iterace č. 4

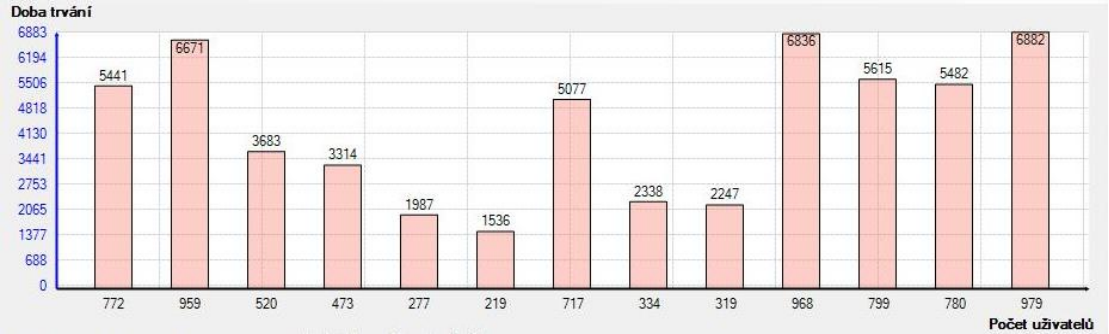
Iterace č. 5

Výstupní grafy z testovací aplikace pro dobu trvání 10 min – AMD 850MHz + 512MB SD RAM:

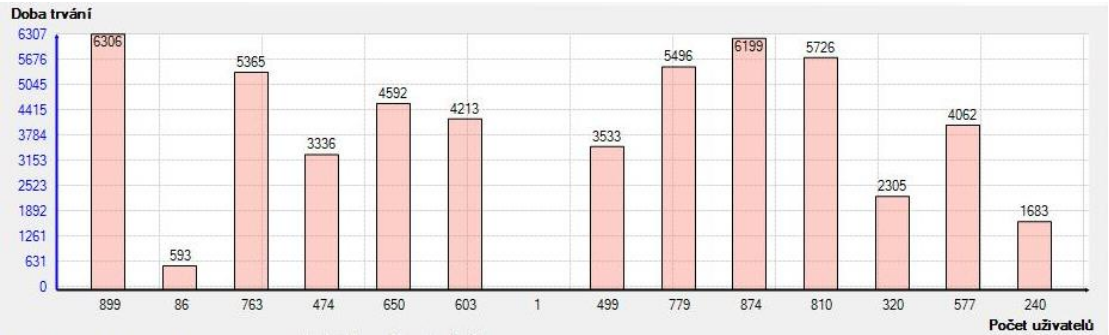
Iterace č. 1



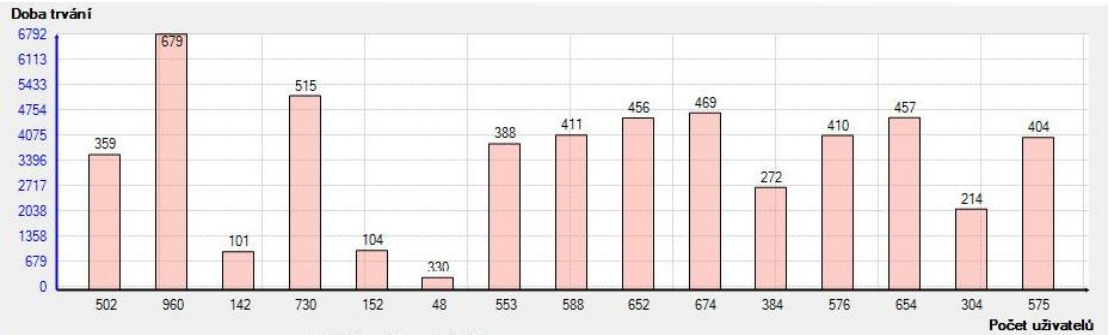
Iterace č. 2



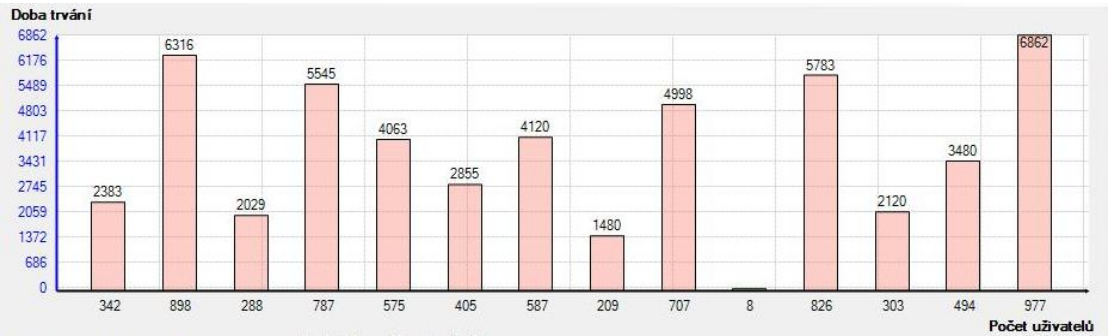
Iterace č. 3



Iterace č. 4



Iterace č. 5



Veškeré ostatní tabulky a výstupy z aplikace jsou přiloženy v disku přiloženém k diplomové práci v souboru Testování_tabulkyAGrafy.xlsx